THESIS

NEXT GENERATION SOFTWARE PROCESS IMPROVEMENT

by

Daniel Turnas

June 2003

Thesis Advisor:	Mikhail Auguston
Second Reader:	Christopher D. Miles

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Next Generation Software Process Improvement			**5. FUNDING NUMBERS**
6. AUTHOR(S): Daniel Turnas			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

Software is often developed under a process that can at best be described as ad hoc. While it is possible to develop quality software under an ad hoc process, formal processes can be developed to help increase the overall quality of the software under development. The application of these processes allows for an organization to mature. The software maturity level, and process improvement, of an organization can be measured with the Capability Maturity Model. The scope of this work is to use organizationally improved software processes on a small scale software product developed by the U.S. Army . The goal is to establish process improvement based on the Capability Maturity Model.

14. SUBJECT TERMS CMM, process improvement, software engineering, requirements management, risk management, software design.			**15. NUMBER OF PAGES** 143
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

NEXT GENERATION SOFTWARE PROCESS IMPROVEMENT

Daniel Turnas
Civilian, United States Army TACOM
B.S., University of Michigan, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2003

Author:	Daniel Turnas
Approved by:	Mikhail Auguston Thesis Advisor
	Christopher D. Miles Second Reader
	Peter Denning Chairman, Computer Science Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Software is often developed under a process that can at best be described as ad hoc. While it is possible to develop quality software under an ad hoc process, formal processes can be developed to help increase the overall quality of the software under development. The application of these processes allows for an organization to mature. The software maturity level, and process improvement, of an organization can be measured with the Capability Maturity Model. The scope of this work is to use organizationally improved software processes on a small scale software product developed by the U.S. Army. The goal is to establish process improvement based on the Capability Maturity Model.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Appreciation to United Media for reprint approval for the appearance of Dilbert in this thesis.

Special thanks to Joseph P. Szafranski. His knowledge on CMM and process improvement provided invaluable reference to this thesis.

Thanks to the SDIP development team, and all those who helped in the development process:
Christopher Ostrowski
John Bohn
Douglas Gersky
Russell Menko
Karen Lafond
Joseph P. Szafranski

Appreciation to all those who contributed their views and comments on the content of this thesis.

Special personal thanks to the late Michael S. Saboe, Ph. D. Without his guidance and foresight, this thesis would not have been possible.

Naval Postgraduate School Advisors
Mikhail Auguston

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. NEXT GENERATION SOFTWARE ENGINEERING TECHNOLOGY AREA

The Next Generation Software Engineering Technology Area (Next Gen) is part of the U.S. Army Tank-automotive Armaments Command (TACOM). The primary mission of Next Gen is to provide software support for the U.S. Army. The software that is developed and managed at Next Gen is used in many of the current fighting vehicles, including the Abrams Main Battle Tank, the Bradley Armored Personnel Carrier, the Wolverine, and the Future Combat Systems. Next Gen is a key player in delivering large scale software solutions for Army systems.

Dedicated to process improvement, Next Gen is constantly looking to improve the organization. One of the key tools in the dedication towards process improvement is the use of the Software Engineering Institute's Capability Maturity Model (CMM) [1]. The CMM is a tool to rate the process maturity level of an organization. In August 2001, Next Gen was certified at CMM for Software Level 3. The certification demonstrates the organization's ability to define and follow organizational policies, procedures, and practices. Though Next Gen was certified at Level 3, there were still some opportunities for improvement within Level 3. The main area involved improving metrics collection processes.

In an effort to improve Level 3 processes and progress towards Level 4, Next Gen began a small scale software project. The Skinny Driver's Instrument Panel was proposed as a means to improve Level 3 processes and begin implementing CMM Level 4 processes. The scope of this thesis is developing a software product based on CMM Level 3 and 4 processes.

B. PROCESS IMPROVEMENT THROUGH CMM

Software development companies are constantly looking to improve the quality of their products. This is often done by defining company processes such as thorough software testing and risk management. The more mature an organization is, the more defined - and ultimately refined - a collection of software development processes is. In order to have a benchmark against which to measure the level of software development maturity, the Software Engineering Institute established the Capability Maturity Model (CMM) [1]. The CMM framework has been applied to focus on different organizational practices. The Capability Maturity Model for Software (SW-CMM) is used to measure the software development maturity level of an organization. The SW-CMM is divided into five maturity levels, ranging from Level 1 – Ad hoc to Level 5 – Optimizing. Each maturity level consists of different Key Practice Areas (KPAs). The different KPAs describe different activities defined and established within each maturity level. As an organization advances in maturity new processes are adopted, as well as the continuation of performing processes defined from lower maturity levels.

The current effort is to integrate the different CMM concepts into the Capability Maturity Model Integration (CMMI). At the time of this thesis, however, Next Gen was still basing process improvement on the SW-CMM Version 1.1.

II. BACKGROUND INFORMATION

A. NEXT GEN BACKGROUND

The Next Generation Software Engineering Technology Area (Next Gen) is part of the U.S. Army Tank-automotive Armaments Command (TACOM). The primary mission of Next Gen is to provide software support for the U.S. Army. The software that is developed at Next Gen is used in many of the current fighting vehicles, including the Abrams Main Battle Tank and the Bradley Armored Personnel Carrier. Established in 1994, Next Gen quickly became a key player in the development of army critical software. While the associates were skilled developers, it was determined that the processes Next Gen followed needed to be improved upon.

While supporting the current fighting forces of the U.S. Army, Next Gen is dedicated to adopting cutting edge processes to enhance the software that drives the United States Army. By adapting software engineering methodologies, Next Gen was able to improve upon its software development processes. This can be seen through the use of the Software Engineering Institute's Capability Maturity Model for Software (SW-CMM).

Discussed in detail below, the SW-CMM is a measurement of an organization's maturity level in developing software products. In August 2001, Next Gen was certified at SW-CMM Level 3. The certification demonstrates the organization's ability to perform software engineering processes. Obtaining SW-CMM Level 3 was not the ultimate goal of process improvement at Next Gen. As seen in [2], the return on investment of transitioning from Level 3 to Level 4 was seen to be 109%. While this may be just one example, it indicates the benefits of continuous process improvement.

Currently certified under SW-CMM Level 3, Next Gen adheres to organizationally established processes in software management, intergroup coordination, organizational process definition, software engineering, and the use of peer reviews. Certification at Level 3 also entails continuing Level 2 processes. Though described in detail below, a brief description is needed to show the path towards process improvement. The processes currently defined and followed at Next Gen include a fully defined software development plan, risk management processes, software engineering processes, and organization processes. The definition and adherence to established processes results in a software development lifecycle where each of the phases is completely defined. The entry and exit requirements, as defined by the organization, are completely understood. Lifecycle metrics, described in detail below, are taken throughout the entire project . These metrics collectively show the status of the entire project.

To transition from SW-CMM Level 3 to Level 4, two key elements have to be integrated into the processes of the organization. The first key element is management based on metrics established at the SW-CMM Level 3. The second key element establishes a quantitative representation of software quality based on metrics established at the SW-CMM Level 3. More simply put, a Level 4 organization will make productive use of collected measurements to improve product and process within an organization.

The most fundamental problem of this thesis describes the need for Next Gen to improve software development processes from SW-CMM Level 3 to Level 4. Included in this effort would be the Level 3 processes identified as opportunities for improvement, mainly Level 3 metrics-based activities. The second problem can be seen within the software development experience of Next Gen. While Next Gen has experience delivering large scale software solutions to the Army, there is little organizational experience developing small scale software solutions. Therefore, initiating SW-CMM Level 4 processes with current projects would require large scale organizational changes.

The proposed solution to both of these problems is the scope of this thesis. This thesis represents the development of a small scale software product to be developed using SW-CMM Level 4 processes. This will help begin the transition from Level 3 to Level 4, and give the organization experience developing small scale software solutions. This thesis will describe the process improvement effort on a small scale software product, from the point of view a software developer.

B. CAPABILITY MATURITY MODEL

Many software organizations implement software engineering practices. The degree to which those methods are implemented can be measured against established models. One such model is the Capability Maturity Model (CMM). The CMM is a model that can be used to measure the maturity to which different organizational procedures are implemented [3]. Developed at the Software Engineering Institute at Carnegie Mellon, the CMM concept has been extended to many different models. Software Acquisition, People, and Software are examples of three different extensions of the CMM concept. Each of the versions of CMM follows the same structure, but is tailored to the specific functional area being performed. This thesis will focus on the CMM for Software (SW-CMM). Discussed below are the different levels of SW-CMM, the Key Process Areas within each level of maturity, and the Key Practices within each Key Process Area.

1. CMM Structure

The SW-CMM can be used to measure the software development maturity level of an organization. The SW-CMM, and each of the other models of CMM, is divided into five levels of maturity. Figure 1, as seen in [1], shows the five levels of CMM. Each level of maturity describes a different set of established activities. The five levels of maturity are Initial, Repeatable, Defined, Managed, and Optimizing. The least mature level of software development, as described by the SW-CMM, is Level 1 - Initial. Level 1 is best described as ad hoc. There are no organizational policies established, or followed, on how to develop a software product. The second level of maturity is the Level 2 - Repeatable. Level 2 describes an organization that follows specific activities along the development process. Another organization at the US Army TACOM just received certification at the CMM for Software Acquisition Level 2 [4]. Level 3 - Defined describes an organization that follows defined processes to progress through each activity. Level 4 of the SW-CMM, Managed, describes an organization that is managed based on measurements taken against the practices established at Level 3. The highest level of maturity, Optimizing, describes an organization that continually improves upon organizational software development activities. Maturing through the CMM levels means acquiring new processes in addition to continuing processes defined at lower levels.

The main benefit of the CMM is to clearly establish what characteristics are necessary for each level, and how to improve an organization's software maturity level [5]. Another strength of the CMM is that it is flexible. The model does not specify what engineering standards an organization must adopt; just that policies, standards, and procedures are defined and followed. As seen in [6], the vast number of software standards mandates that a model used to determine maturity must not be based on one particular standard.

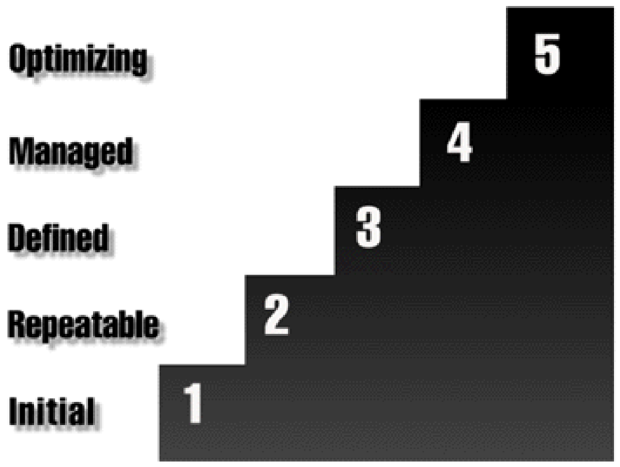


Figure 1. CMM levels (From: [1])

a. Key Process Areas

The SW-CMM is structured to determine the software development maturity level of an organization. This can be done by determining the activities that an organization performs throughout the development lifecycle. To eliminate confusion, different activities are localized to an individual maturity level. This will better help map an organization's performance as described by the SW-CMM.

Within each SW-CMM maturity level Key Process Areas (KPAs) are defined. The KPAs describe the activities that are needed to be performed for an organization to obtain a certain maturity level [7]. Level 1, for example, has no KPAs defined. This coincides with the fact that the Initial Level is ad hoc, with few if any established policies. Level 2 KPAs include requirements management, configuration management, and project planning. Level 3 KPAs include integrated software management, software product engineering, and peer reviews. These KPAs establish that a Level 3 organization follows defined processes on performing Level 2 activities. Level 4 KPAs are quantitative process management and software quality management. Level 5 KPAs are defect prevention and process change management.

b. Key Practices

To help determine which KPAs are being implemented by an organization, Key Practices (KPs) are defined. The SW-CMM describes several KPs that are the specific tasks necessary to show that each KPA is implemented within an organization. A Level 2 KP, for example, is "The software engineering group participates on the project proposal team" [1]. Each KP can be traced to a specific organization document or defined procedure that is used to show implementation. Each software development project will show examples of performing the specific KP by a specific project artifact or process implementation.

The focus of this thesis is to establish the SW-CMM Level 3 and Level 4 processes performed within Next Gen as seen on a small scale software product. This thesis, therefore, will focus the KPAs and KPs within Level 3 and Level 4. Because this thesis describes the work performed normally associated with a software developer, only the KPAs and KPs normally encountered from a developer's point of view will be presented. The following sections describe the Level 3 and Level 4 KPAs and KPs that will be focused on within this thesis.

2. SW-CMM Level 3

An organization that is certified at the SW-CMM Level 3 has defined software development processes. These processes are defined in policies established at the management, organizational, and engineering level. Management processes include integrated software management and intergroup coordination. Organizational processes include organization process focus, organization process definition, and a training program. Engineering processes include software product engineering and the use of peer reviews. The management, organization and engineering processes map directly to SW-CMM KPAs. The focus of this paper will reflect the work that was done towards implementing a potential solution within Next Gen. The KPAs discussed below, therefore, will be Integrated Software Management, Software Product Engineering, and Peer Reviews. Metrics collected within each KPA are also required within Level 3 activities. Metrics will also be described in detail below.

a. Integrated Software Management KPA

The first SW-CMM Level 3 KPA as seen from the developer's point of view is Integrated Software Management. This Key Process Area describes activities including estimating, planning and risk management. These activities represent what is required above and beyond normal software development activities. Each of these activities is described below.

1. Estimating. The first major activity within the Integrated Software Management KPA is estimation. Project development starts with estimating. The two main items that are estimated are cost and schedule [8]. Based on the cost and schedule estimates, resources required to develop the product and scope of the product can be determined. The cost can be determined based on source lines of code. There are, however, disadvantages to estimating based on code size. The programming environment, efficiency of the developers, and the availability of software reuse all have a variable affect on the amount of code needed for the product. The most effective means

to estimating cost is to combine several different estimating techniques. Combining estimates on code size, developer experience, and function point analysis will result in a more reliable estimated cost. While estimating cost can be somewhat difficult, estimating schedule is more straightforward. The schedule is usually driven by the needs of the stakeholders or the resources of the developers. Once estimates are determined, they can be periodically compared to actual schedule and cost measurements throughout the project lifecycle.

2. Planning. The second major activity within Integrated Software Management is planning. Planning is achieved through the use of lifecycle models. The lifecycle model establishes the major project phases, the milestones, and what activities must be performed within each phase. Among the major models are waterfall, spiral, and a combination of incremental and iterative. The first model to be adopted was the waterfall. Each phase of the project was dependent on the completion of the previous phase. Since no phase could be revisited, several drawbacks to this model lead to the development of the spiral model. A major component of the spiral model is the use of prototyping. Early prototypes helped the developers and stakeholders come to an agreement on requirements. This model also had several disadvantages. One of the biggest drawbacks is that the stakeholders would end up focusing on what they could see and not the true functionality of the product. Since the prototype was most useful in showing interfaces, the spiral model lead to major revisions of requirements for only the interfaces. The most recent lifecycle model to be used is a hybrid incremental and iterative model. This lifecycle model allows for the developers to apply the divide and conquer concept, all while incorporating feedback from stakeholders through incremental review of the product.

3. Risk management. In a perfect world, a software product can be developed that exceeds the stakeholder needs and be delivered ahead of schedule. Of course, this is not a perfect world, and obstacles arise that stand in the way of software development. Project leaders must address the probability that risks may occur by implementing risk management, the third major activity of Integrated Software

Management. Risk management is the process of being aware of potential risks and determining a course of action to prevent or recover from any risk that may occur [9]. Risk management is composed of two major items: risk assessment and risk control [10].

The first item of risk management is determining a list of potential risks. Risks are events that can happen that have a negative effect on developing a product. Once the risks have been identified, each is analyzed for probability of occurring and severity of the consequences. Based on this information, a mitigation technique is determined. The technique may be to prevent the risk from occurring or to minimize the consequences once the risk occurs.

The second item in risk management is tracking potential risks. Tracking risks requires a constant awareness of project development status. To keep track of project status, metrics are collected at regular intervals. Metrics that are normally collected are coding status, funding levels, and software defects [11]. These and other metrics give an accurate status of possible risks. By identifying, preparing for, and monitoring possible risks, a software development project can mitigate many of the unforeseen obstacles that may occur.

b. Software Product Engineering KPA

The second SW-CMM Level 3 KPA typically encountered by a software developer is the Software Product Engineering KPA. This Key Process Area describes activities normally associate with developing software. Activities within this KPA include requirements management, developing software designs, creating the software, and testing the final software product. Each of these activities is described below.

1. Requirements Management. Of all the responsibilities that a development team must undertake, requirements management may be the most important. Requirements management, the first activity of Software Product Engineering, can be broken down into two main components: requirements gathering and requirements tracking. Gathering the requirements must include an exhaustive interviewing of all the stakeholders to understand what the product must do and the

environment under which it must perform [12]. Without a complete and thorough collection of requirements, the product will likely fail to meet the needs of the stakeholders. However, even with a complete understanding of the product requirements, the product could still be developed incorrectly. See Figure 2 as an example. The solution to this problem is the second component of requirements management - requirements traceability. This component of requirements management consists of tracking each individual requirement. Tracking starts from the point that it was acquired from a stakeholder interview and ends when it is tested before the complete product can be delivered [13]. The critical aspect in requirements traceability is the ability to handle the large number of requirements that even the most trivial software product is composed of. Both components of requirements management help to assure that the developers know the needed characteristics of the product and continue to develop the right product throughout the entire development lifecycle.

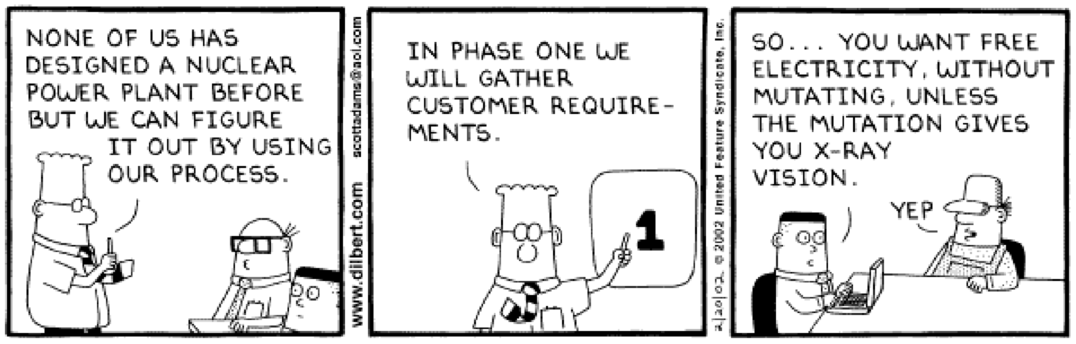

NONE OF US HAS DESIGNED A NUCLEAR POWER PLANT BEFORE BUT WE CAN FIGURE IT OUT BY USING OUR PROCESS.
IN PHASE ONE WE WILL GATHER CUSTOMER REQUIREMENTS.
1
SO... YOU WANT FREE ELECTRICITY, WITHOUT MUTATING, UNLESS THE MUTATION GIVES YOU X-RAY VISION.
YEP
www.dilbert.com scottadams@aol.com
© 2002 United Feature Syndicate, Inc.

Figure 2. Real world requirements management

2. Software Design. All of the Key Practices encountered so far focus on what the product needs to do. The design phase allows the developers their first opportunity to address how the product will accomplish these needs. Proper requirements management enables the development team to work with a fixed set of requirements. How those requirements are to be implemented is determined by the software design [14]. Software design, the second major activity in Software Product Engineering, focuses on creating an architecture of the product to be developed. The software architecture is a blueprint of how the system comes together as a whole. An

architecture is an overall structure of a system and is composed of many different perspectives. Each of these perspectives represents a view of the system. Some of the different views include hardware, software, and interfaces. Each of the perspectives starts at a high level of abstraction. Each time the perspective is shown in a diagram, the amount of abstraction is decreased until a high enough level of detail exists. These diagrams are then used as the basis for the development. The designed architecture will guide the developers on how the system is to be developed.

Software design is a new and advancing discipline. New methods and concepts are being developed to advance the field. One of the most influential advancement comes in the form of object oriented design. Object oriented design focuses on the use of abstraction to simplify the representation of systems [15]. Diagrams show an abstract representation to better communicate than through the use of potentially ambiguous text. Systems can be represented as a collection of subsystems, and subsystems can be even further decomposed. At the most detailed level, the diagrams are composed of individual objects. Similarly, a system can be represented as a collection of classes, where each class has key characteristics. The class diagram can be decomposed into more detailed classes, with more specific information. The final level of decomposition is a collection of individual objects [16].

Abstraction allows the system under development to be represented graphically. With the goal of a common graphical notation, Unified Modeling Language (UML) was developed. UML is a set of standards on how to graphically represent a system [17]. Based on its many strengths, UML has become one of the most widely accepted standards to use while designing object oriented systems. The modeling language is used to standardize the diagrams used in object oriented design.

While object oriented design is prominent in the software engineering field, agent oriented design is emerging. Agent oriented design, based on object oriented technologies, shifts the focus from a data-centric view towards a processcentric view. One of the most advanced agent oriented architectures is Cougaar [18]. Based on the human cognitive process, Cougaar performs three main functions. The first function is to decompose larger tasks into smaller tasks. The second function is to allocate resources to those smaller tasks. The third function is to continually assess the smaller tasks.

3. Testing. Testing, the third activity of Software Product Engineering, is the final phase of software development. After the software product is developed based on the system architecture, a rigorous series of tests must be performed. These tests are performed by unbiased testers and used to verify that the product was developed correctly. The testing is done in accordance with the Software Test Plan. Most common testing techniques include unit, integration, white-box, and black-box testing [19]. Unit testing is done at the subsystem level before it is integrated into the system as a whole. The subsystem can be tested without taking into account any influences from other subsystems. Once the subsystem passes unit testing, the product undergoes integration testing. This phase tests all of the subsystems and how they interface with each other. Unit testing and integration testing can be done with knowledge of how the system of subsystem works. White-box testing uses this knowledge by testing the inner functions of the systems or subsystems. Black-box testing, on the other hand, simply tests inputs and outputs of the system or subsystem being testing. A combination of white-box and black-box testing results in a more thoroughly tested software product than a product that was tested using only one of the techniques. While only a trivial product can be exhaustively tested, automated testing tools can help the testing process to be more thorough [20]. A product that passes carefully tested procedures is ready for delivery to the stakeholders.

The Integrated Software Management and Software Product Engineering KPA describe the major project activities. These activities are performed based on procedures outlined in several lifecycle documents, such as a software development plan, software requirements specification, and software test plan. These major program documents establish the basis for how the product will be developed, what the product will do, and how the product will be tested. The Software Development Plan presents an overview of the project development. It discusses project estimates, the schedule, lifecycle model, what artifacts will be created, and when and how the project will progress through the entire lifecycle. The Software Requirements Specifications document is the document that will be used for requirements signoff by the stakeholders. It will be used as a bases of requirements management, as described above. The Software Test Plan addresses how the product will be tested. The specific test procedures will be presented in another lifecycle artifact, but the overall testing strategy is the scope of the Software Test Plan. Once the lifecycle documents have been written, the development team has a roadmap of the entire development process.

c. Peer Reviews KPA

The third SW-CMM Level 3 KPA this thesis focuses on is Peer Reviews. From the developer's point of view, peer reviews are an integral and ongoing part of an organization certified at Level 3. Peer reviews, review meetings held with the developer's peers, are used throughout the entire lifecycle to systematically review software work products for defects or areas that need changes. Peer reviews use the testing technique of inspection. Inspection, as seen in [21], is a method for identifying defects in a software work product. Once a work product passes a peer review, it is considered by an organization as complete. While there are other means for eliminating defects in a software work product, inspection is a simple and effective way for an organization to assure the quality of that work product. An increase in development time of only 15% is required for, but peer reviews ultimately eliminate time spent on correcting defects found later in the project lifecycle [22]. The Peer Review KPA activities include: conducting peer reviews according to organizational policies, recording data collected from peer reviews, and having a software quality assurance representative participate in peer reviews.

d. Metrics

While "metrics" is not a specific KPA, the use of metrics appears throughout all Level 3 KPAs. Metrics is discussed in a separate section based on a Next Gen organization need. During the SW-CMM Level 3 certification at Next Gen, metrics-based processes were identified as an opportunity for improvement.

Metrics are quantitative measures of a software process or product. Metrics can include defects per lines of code, peer review duration, number of pages of a document, or simply the number of lines of code. The collection and review of metrics provides a numerical view of a project's status. Based on these metrics, actions can be taken if needed. As seen in [23], the use of metrics at Bull's Enterprise Servers Operation improved the overall quality of project management. Improvements in software product and organizational processes can also be made by collecting and analyzing metrics.

One major concern with the use of metrics is what to measure. The Goal-Question-Metric (GQM) approach can be used to determine which metrics are to be collected [24]. The GQM is a three step approach. The first step is listing the goals of the organization. The second step is listing questions that can be asked, whose answers determine if the goals have been met. The final step is to list metrics that can be used to answer the questions listed in step two. As described in [25], another way to determine which metrics to use can be determined by the process maturity level of an organization. For example, if peer reviews are conducted then peer review metrics are appropriate. Once a software development lifecycle is completed, an organization can review collected metrics and determine usefulness. This can be used when determining what metrics to collect on future software projects.

Once collected, the data must be analyzed. The use of metrics collection tools can aid in collection and analysis. The metrics collection tools used must be based on the metrics collected [26]. The information obtained from collected metrics can then be used in evaluating activities throughout SW-CMM Level 3.

3. SW-CMM Level 4

An organization that is certified at the SW-CMM Level 4 can be described as metrics managed. Both product and process are quantitatively managed based on metrics collected during SW-CMM Level 3 activities. Progress can be compared numerically against estimates and plans established at the start of the project lifecycle. As seen in [27], analysis of metrics, or multiple metrics, can pinpoint problems in individual products or in the process itself. With this knowledge, a Level 4 organization can take corrective actions to address any issues that analysis of metrics pinpoints. The periodic evaluation of selected metrics should be performed. As seen in [28], the usability of selected metrics should be periodically reviewed. This can result in metrics giving a better representation of an organization. The organization will be better informed to make metrics-based decisions. The Key Process Areas defined at Level 4 are Quantitative Process Management and Software Quality Management. Both of the SW-CMM Level 4 KPAs are within the scope of this thesis, and will be discussed below.

a. Quantitative Process Management

The first Level 4 KPA is focused on managing processes based on metrics. As seen in [29], evolving through the SW-CMM levels increases process productivity levels in schedule, effort, and reliability. These increases can be achieved through the Quantitative Process Management KPA. This KPA is composed of several key practices. An organization must follow an established process for managing projects based on metrics. In order to help perform this KPA, a group is established to coordinate quantitative process management activities. In order to fully make use of metrics, support must be in place to collect and analyze project metrics. Once analyzed, the information is documented and distributed. Management decisions can then be made based on process metrics collected and analyzed.

b. Software Quality Management

While the first Level 4 KPA focuses on process metrics management, the second Level 4 KPA focuses on product metrics management. The Software Quality Management KPA establishes key practices that allow the quality of the software being developed to be managed. Like the first KPA, the second Level 4 KPA requires that metrics-based management activities be planned, this time concerning product quality issues. The project must also have defined measurable goals for product quality, see [30]. These goals can be derived from how well the organization performs the SW-CMM Level 3 Software Product Engineering KPA. Measurements based on the results of requirements management, design, coding, and testing can be used in product metrics-based management. Software quality assurance (SQA) is a significant component of the Software Quality Management. As seen in [31], SQA is "a systematic effort to improve the delivery condition [of the software product or process]." One of the roles of SQA is performing software product audits [32]. These audits help verify the accuracy of metrics used in the Software Quality Management KPA.

4. Summary

The Software Engineering Institute has developed a model to determine an organization's software development maturity level. The Capability Maturity Model for Software is a five level model. Each step is composed of increasingly mature software engineering processes. Processes improvement can be accomplished by defining, and performing, processes for activities described within each level of the model.

The Next Generation Software Engineering Technology Area is continuously dedicated towards process improvement. Recent certification at SW-CMM Level 3 is just one milestone along the path of process improvement. To begin the effort required for certification at SW-CMM Level 4, Next Gen needed to improve some of the Level 3 activities identified as opportunities for improvement. Most notably, these are metrics-based activities.

Due to the large scale nature of the products developed within Next Gen, new processes could not easily implemented. A small scale software product was proposed as a pilot project for process improvement. This project would be developed under improved Level 3 processes, along with newly defined Level 4 processes.

III. PROPOSED SOLUTION

A. SKINNY DRIVER'S INSTRUMENT PANEL

Next Gen is dedicated to the goal of continuous process improvement. Due to the large scale software products normally associated with Next Gen, a small scale project was proposed as a means to implement process improvement activities. The process improvement on a small scale project could then be incorporated into the large scale projects. The level of effort was determined to be suitable as a Naval Postgraduate School thesis project. The project was to have two goals: advancing software engineering technologies and improving software development processes within Next Gen.

The first project goal was for the finished software product to be used to help develop reconfigurable software. The software, along with supporting design documentation, would be delivered to a consortium of government, industry, and academic key players working with reconfigurable software technologies. This consortium was collaborating on a project called Dynamic Assembly for System Adaptability, Dependability, and Assurance (DASADA) [33]. Several potential contributors were proposed for providing solutions for enabling the DASADA technologies. Next Gen, along with other leading government, industry, and academic experts proposed the Dependable Automated Reconfigurable Technology for Software (DARTS). The Next Gen proposal would use the Skinny Driver's Instrument Panel (SDIP) as a test article to help develop the DASADA technologies. A successful DASADA project could potentially affect software technology found in future US Army systems.

The second goal of the SDIP project was to implement processes improvement activities within Next Gen. Recent certification at the Software Engineering Institute's SW-CMM Level 3 was the starting point of process improvement on the SDIP project. The SDIP project would improve on the Level 3 activities, and define Level 4 processes.

Functionally, the SDIP system would be a reduced functionality simulator of the Abrams Main Battle Tank Drivers Independent Display console. A fully functional simulator existed in-house, but could not be release to DASADA/DARTS partners due to security issues. Reducing the functionality of the simulator would result in a non-classified system that could be released to DASADA/DARTS partners. Developers had access to the source code of the existing simulator, but there would be a short lead time to obtaining it.

Structurally, the SDIP system would be composed of three components. The first component would be the reduced functionality driver's display. The second component would be a virtual driver, or autopilot, that provided the driver's display with commands. The third component would be a recreation of the existing Abrams data bus structure. Figure 3, below, shows the SDIP system level design.

Based on the DASADA schedule, the SDIP project would be time-boxed as a six-month project. The development would be done with a three man team.

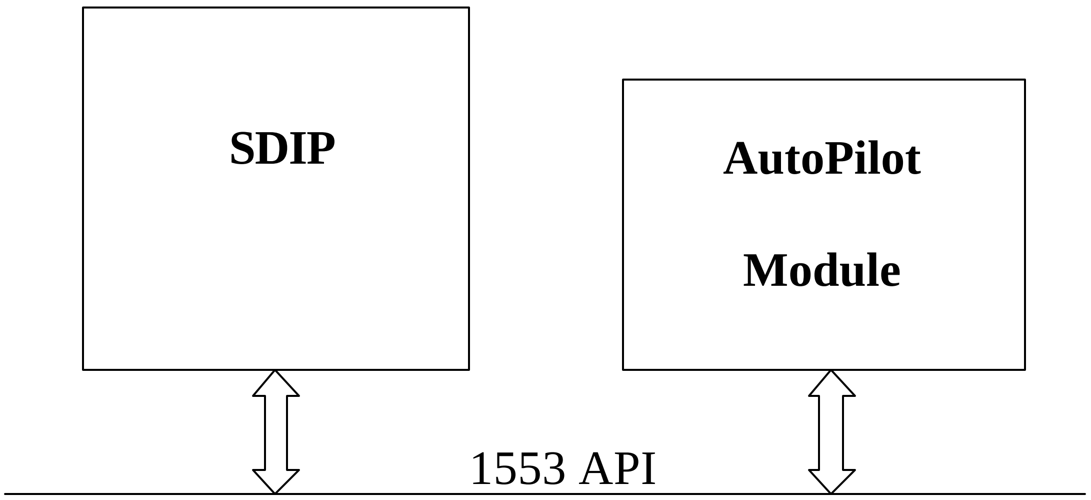


Figure 3. SDIP system diagram

B. IMPLEMENTATION EFFORT

A three man team was allocated to implementing the SDIP system. The author of this thesis and two addition developers were to perform all SDIP development activities per Next Gen defined processes. Each developer was pivotal in performing all SW-CMM Level 3 activities. The work performed by the author of this thesis is described in detail in the Findings chapter. In summary, the following work was performed by this author:

- Performing initial planning, estimating, and risk management activities
- Contributing resources to Process Action Team in metrics definition
- Participating and contributing in peer review activities
- Contributing in regular group meetings and status reviews
- Contributing to metrics collection and analysis
- Providing significant contribution to content of lifecycle documents
- Performing requirements management activities
- Making design decisions and communicating design concepts
- Providing coding assistance in all software components
- Performing complete coding of Autopilot module
- Conducting complete analysis of SDIP process improvement efforts
- Performing normal project activities not specifically mentioned in this thesis

THIS PAGE INTENTIONALLY LEFT BLANK

IV. FINDINGS

A. PERFORMANCE TOWARDS SW-CMM LEVEL 3

While the ultimate goal of the SDIP development project was beginning the transition towards SW-CMM Level 4 certification, the project had to first adhere to Level 3 activities. From a developer's point of view, these activities are seen within the Integrated Software Management, Software Product Engineering, and Peer Review Key Process Areas. These activities would be performed based on the processes verified during the SW-CMM Level 3 certification process.

The CMM Level 3 goal of the SDIP development project was to incorporate the opportunities for improvements cited during the SW-CMM Level 3 certification process. The main opportunity for improvement at the Level 3 involves establishing more thorough metrics-based development activities. Metrics collected on previous projects tended to be obtained near the end of the development lifecycle. This resulted in metrics that did not have significant value added to the project. A more detailed process concerning metrics was the Level 3 goal of the SDIP development project.

1. Integrated Software Management

a. Estimating

The first major activity within the Integrated Software Management KPA is estimating. Estimating was performed by the SDIP development team. The initial project kickoff email, see Appendix A, requested estimates on a schedule and plan, which the development team provided.

The estimation of the development projected consisted of three categories: software size, estimated man-hours, and documentation required. The software size estimates, see Table 1, would be used to determine the required workload. Once the workload was determined, it could be used to affect the scope of the project. The estimated workload, see Table 2 for estimates, was consistent with the manpower that was assigned to the project, so the scope remained as established in the project kickoff email.

SDIP Computer Software Components	**KLOC**
SDIP CSC	1.5
Autopilot CSC	0.5
1553 Interface CSC	1.5
Total	3.5

Table 1. SDIP code size estimates

LABOR CATEGORIES	**ESTIMATES (HRS)**
Planning	176
Tracking	120
Reviews and Audits	80
Analysis	152
Requirements	240
Design/Code	1104
Testing	40
SQA	394
SCM	40
Debriefing	8
TOTALS	**2354**

Table 2. Man-hour estimates for major activities

The third category of project estimation was based on the documentation required. During creation of the software development plan, an initial list of lifecycle artifacts was created. Based on this list it was determined that additional resources were needed to complete the scope of the project. Resources were allocated for configuration management, quality assurance, testing, and systems engineering. The additional resources were allocated based on the following list of estimated lifecycle artifacts:

Software Development Plan (SDP)

Software Requirements Specification (SRS)

Software Design Document (SDD)

Software Configuration Management Plan (SCMP)

Software Quality Assurance Plan (SQAP)

Software Test Plan (STP)

Software Test Coverage Outline

Software Test Cases

Lessons Learned Report

SDIP CSC (code)

Autopilot CSC (code)

1553 Interface CSC (code)

User's Guide and Installation Manual

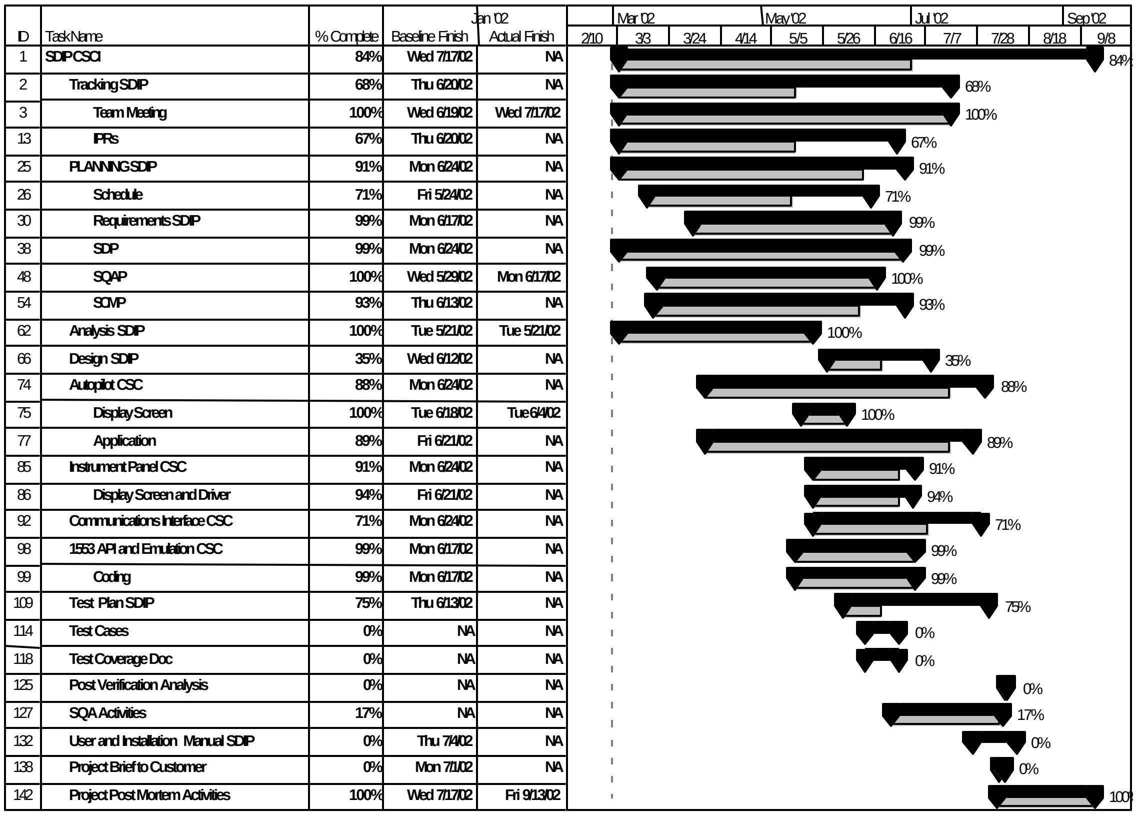
b. Planning

The second major activity within the Integrated Software Management KPA performed by the SDIP team was project planning. As seen in [34], determining the project schedule is difficult. The SDIP project, however, did not have this problem. The project was time-boxed based on DASADA requirements, so creating the schedule consisted of allocating the given time to the different lifecycle phases. The different phases were assessed for required effort, and the available resources were used to determine the amount of time to allocate to each phase. Each phase was analyzed to determine the required lifecycle artifacts, and the schedule was created. Though the schedule was never base-lined until late in the lifecycle, the majority of the original schedule remained the same. The planned schedule can be seen in Figure 4.

Based on initial project requirements, the SDIP development effort was to follow an iterative and incremental lifecycle model. An early prototype was necessary to demonstrate functionality. This required the development team to adopt a incremental approach. Based on the aggressive schedule, an iterative approach was also necessary. This would allow the development team to implement functionality under the time constraints. Even if the project was not completed by the scheduled date, completed iterations could be delivered. Each iteration would undergo a complete plan, design, code, and test phase for the scope of that iteration. The iteration that was completed by the delivery date could still produce an operational DASADA test article.

The iterative and incremental hybrid lifecycle model underwent traditional lifecycle phases. The first phase the developers encountered was the project planning phase. As seen in the schedule below, the schedule was established and the project officially began. The first phase consisted of Integrated Software Management activities, including planning and risk management. The second phase of the SDIP development was the design phase. The developers broke down the system into computer software components and obtained designs for each component. The third phase the development team worked on was the coding phase. Each computer software component was developed and integrated into the system. By the project cancellation date, the software demonstrated functionality consistent with the majority of the initial requirements. Due to the project cancellation, however, the two final phases of the planned lifecycle model were not performed. The limited testing that occurred was only informal debug testing done by individual developers. Because the project was not completed based on funding decisions, there was no maintenance phase encountered.

The project planning was represented in the Software Development Plan (SDP) document, see Appendix B. The SDP contained several sections describing metrics-based activities to be performed. This led to a problem completing the planning phase. Because significant effort was allocated for determining which metrics were to be collected and how, these sections of the SDP could not be completed on time. Multiple revisions to these sections prevented the planning phase to be completed on schedule. Authority was given to being the design phase even though management had not approved a final SDP. It was determined that the concepts were finalized and just minor revisions were necessary for the SDP to be completed. The SDP was finally approved late in the project lifecycle.



ID	Task Name	% Complete	Baseline Finish	Actual Finish
1	SDIP CSCI	84%	Wed 7/17/02	NA
2	Tracking SDIP	68%	Thu 6/20/02	NA
3	Team Meeting	100%	Wed 6/19/02	Wed 7/17/02
13	IPRs	67%	Thu 6/20/02	NA
25	PLANNING SDIP	91%	Mon 6/24/02	NA
26	Schedule	71%	Fri 5/24/02	NA
30	Requirements SDIP	99%	Mon 6/17/02	NA
38	SDP	99%	Mon 6/24/02	NA
48	SQAP	100%	Wed 5/29/02	Mon 6/17/02
54	SCMP	93%	Thu 6/13/02	NA
62	Analysis SDIP	100%	Tue 5/21/02	Tue 5/21/02
66	Design SDIP	35%	Wed 6/12/02	NA
74	Autopilot CSC	88%	Mon 6/24/02	NA
75	Display Screen	100%	Tue 6/18/02	Tue 6/4/02
77	Application	89%	Fri 6/21/02	NA
85	Instrument Panel CSC	91%	Mon 6/24/02	NA
86	Display Screen and Driver	94%	Fri 6/21/02	NA
92	Communications Interface CSC	71%	Mon 6/24/02	NA
98	1553 API and Emulation CSC	99%	Mon 6/17/02	NA
99	Coding	99%	Mon 6/17/02	NA
109	Test Plan SDIP	75%	Thu 6/13/02	NA
114	Test Cases	0%	NA	NA
118	Test Coverage Doc	0%	NA	NA
125	Post Verification Analysis	0%	NA	NA
127	SQA Activities	17%	NA	NA
132	User and Installation Manual SDIP	0%	Thu 7/4/02	NA
138	Project Brief to Customer	0%	Mon 7/1/02	NA
142	Project Post Mortem Activities	100%	Wed 7/17/02	Fri 9/13/02

Figure 4. Estimated schedule with actual completion dates

c. *Risk Management*

The third major activity within the Integrated Software Management KPA is risk management. During the SW-CMM Level 3 certification, risk management was identified as an opportunity for improvement. Before the SDIP project, risk management was an ad hoc process. Risk management consisted only of presenting risk status at monthly In Process Review (IPR) meetings. The SDIP development team established a process for risk management. The Software Development Plan defined a more mature risk management process. Risk management was defined within four basic categories: risk mitigation, risk tracking, configuration management, and quality assurance.

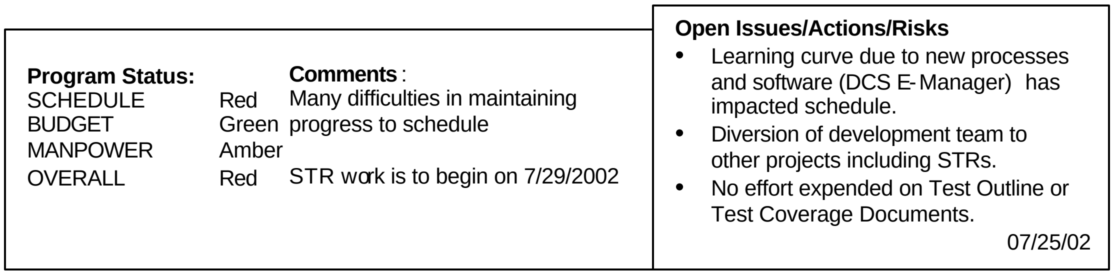
1. Risk Assessment. The first component of risk management that was performed based on defined processes was risk assessment. During the project planning the SDIP development team performed risk assessment. Resources were consulted to add quality assurance and software engineering process perspectives on risk management. Risk identification was performed on all aspects of the development lifecycle. Once the risks were identified, each was given a severity and probability. The risk level was calculated and an appropriate mitigation strategy was identified. A total of twelve major risks were identified. Table 3 shows a portion of the risk assessment table that was developed for the SDIP system. The complete risk assessment table can be seen in Appendix B.

Risk	**Severity**	**Probability**	**Risk Level**	**Mitigation Strategy**
Diversion of development team to other projects including GWT.	High	High	High	Keep management and customer up to date on status.
Inability to meet project goals due to inexperienced development team.	High	High	High	Make extensive use of Next Gen and DCS domain experts and NPS course work.
Requirements creep	Medium	Low	Low	Requirements management

Table 3. Partial risk assessment matrix

During the project lifecycle, there were two major risks that were encountered that were not identified in the risk assessment matrix. The first risk that occurred was the delivery of original source code that was not usable by the SDIP development team. This lead to the second risk identified in the risk assessment matrix, of development being behind schedule. The second risk was the cancellation of the DASADA/DARTS proposal.

2. Risk Tracking. The second component of risk management that was performed based on defined processes was risk tracking. Risk tracking was accomplished by collecting and analyzing metrics. Project metrics, described in detail below, allowed the project team to numerically track the progress of the project development against the initial project estimates. This would uncover areas of risk. Once risks were uncovered, they were formally presented during weekly and monthly project meetings. Risk presentation can be seen in Figure 5 below. The risk assessment matrix was updated throughout the project lifecycle. The updates were made, however, on an irregular basis.



Program Status:		**Comments**:
SCHEDULE	Red	Many difficulties in maintaining progress to schedule
BUDGET	Green	
MANPOWER	Amber	
OVERALL	Red	STR work is to begin on 7/29/2002

Open Issues/Actions/Risks

- Learning curve due to new processes and software (DCS E-Manager) has impacted schedule.
- Diversion of development team to other projects including STRs.
- No effort expended on Test Outline or Test Coverage Documents.

07/25/02

Figure 5. Risk status as seen on a monthly briefing chart.

3. Configuration Management. The third component of risk management performed based on defined processes was the use of configuration management. During project planning, the SDIP development team allocated resources for configuration management (CM). The use of CM would help maintain version control of all lifecycle artifacts. The use of CM was a mitigation technique based on the risks associated with multiple versions of documents and source code anticipated during

project development. A configuration management plan was created to establish the role of CM during the SDIP system development. As established by the plan, all documentation would pass peer review and customer signoff before being placed under control. Software would also be placed under control after it passed peer review. The Software Configuration Management Plan can be seen in Appendix D. A breakdown of the configuration management performed throughout the SDIP development lifecycle can be seen in Table 4.

Configuration Controlled Items	Number of Items
Lifecycle Documents	4
Document Revisions	0
Software Source Files	0

Table 4. Items placed under configuration management

4. Quality Assurance. The fourth component of risk management performed based on defined processes was the use of quality assurance. The SDIP development team allocated resources for quality assurance to help perform risk management. A software quality assurance representative was pivotal in helping the development team identify, collect, and track metrics throughout the entire lifecycle. As stated in the Software Quality Assurance Plan, the quality assurance representative participated in project planning activities, metrics tracking activities, risk management, and performing reviews and audits of the processes established by planning documents.

2. Software Product Engineering

a. Requirements Management

The first major activity within the Software Product Engineering KPA is requirements management. During the SW-CMM Level 3 certification, requirements management was identified as an opportunity for improvement. Before the SDIP project, requirements management was an ad hoc process. The SDIP development team established a software requirements specification document. Though a formal requirements management process was not defined for the organization, requirements management activities were defined in the SDIP Software Development Plan.

The SDIP development team performed requirements management based on a defined project process. Requirements collection, tracking, and testing activities were all defined within the Software Development Plan. The requirements were collected from the customer in an initial project kickoff email. Appendix A shows the project kickoff email from the customer. After review of the initial requirements list, the developers met with the customer to discuss more details of the requested requirements. Once an understanding of customer needs was reached, the developers began work on a Software Requirements Specification (SRS) document. The complete SRS can be seen in Appendix C.

The SRS established requirements identification and a means for requirements traceability. Each of the 49 requirements was given a unique identification number and assigned to a single computer software component. This identification number and assignment allowed the developers to assure that each requirement could be implemented in the design documentation. The requirements could then be tracked to a specific test procedure to ensure every requirement was implemented in the final software product.

Two sample requirements as seen in the SRS:

(004) The SDIP CSCI shall maintain the ability to process 1553 data packets in a manner consistent with the actual method used within a M1A2 tank. That is, the SDIP and Autopilot CSCs maintain the M1A2 1553 packet specifications for interprocess communication of data.

(025) The following group of Project Critical data packets shall be implemented:

Data Packet ID	**Data Packet Description**	**Source LRU**	**Destination LRU**
DP0400.2_DEV_PWR_ST	Device Power Status	TEU	DID
DP0600.2_AUTO_ST	Automotive Status	TEU	DID
DP0800.1_NAV_HEADING	Pos/Nav Heading	TEU	DID
DP0900.2_LOW_RATE_NAV_OUTPUT	Pos/Nav Low Rate Data	TEU	DID
DP1800.2_WAYPT_DATA	Waypoint Data	TEU	DID

Table 5. Requirement (025) breakdown.

b. Software Design

The second major activity within the Software Product Engineering KPA is software design. The design activities were the least accomplished activities of the Software Product Engineering KPA. Several factors contributed to the inadequate amount of design that was performed by the development team.

The SDIP development was initially based on the existing Abrams M1A2 system. Based on this information, the development team chose to use existing design documentation to represent the SDIP system. After the existing source code was deemed unusable, the development team had to reevaluate the development plan. With the level of experience of the developers and the aggressive schedule, the development team decided that software could not be developed based on existing design documentation. After management proposed a new development concept for the system it was clear that new design documentation was needed. The new design would be based on similar existing functionality, but be designed with a different architecture. Due to the delay in waiting for the source code, the development team had to make up missed coding time. Formal design activities were replaced with quick informal team meetings to discuss the new architecture.

The original SDIP system consisted of three computer software components (CSCs). The first CSC was the existing driver's display of the M1A2 system. The second CSC was the existing 1553 Data Bus. The third CSC was an autopilot module that would act as a virtual driver. Normal driver commands would be sent to the driver CSC via the 1553 CSC. Because the first two CSCs were based on existing systems, the design for those CSCs were the existing system design documents. Design effort was only allocated to the third CSC. When design effort was halted, only the design for the AutoPilot CSC was documented. Estimated to be only 55% complete, the design consisted of three levels of detail.

The first level of detail, the architectural design, is an abstract view of system. The architectural view shows abstract state diagrams, software components, and use cases. The second level of detail, the mechanistic design, is a more detailed view of the system. The mechanistic design shows detailed use cases, class diagrams, and sample sequence diagrams. The most detailed view of the system is the detailed design. The detailed design shows detailed state diagrams and pseudocode. It was attempted to bring additional resources onto the project to help develop the design documentation. There was not enough resources to help bring the new designers up to speed, so the design documentation was never completed.

1. Architectural Design. The architectural design completed on the SDIP project consisted of a system diagram, a list of objects and software components, use cases, and a class diagram of the AutoPilot component. The AutoPilot control flow can be seen in the Figure 6.

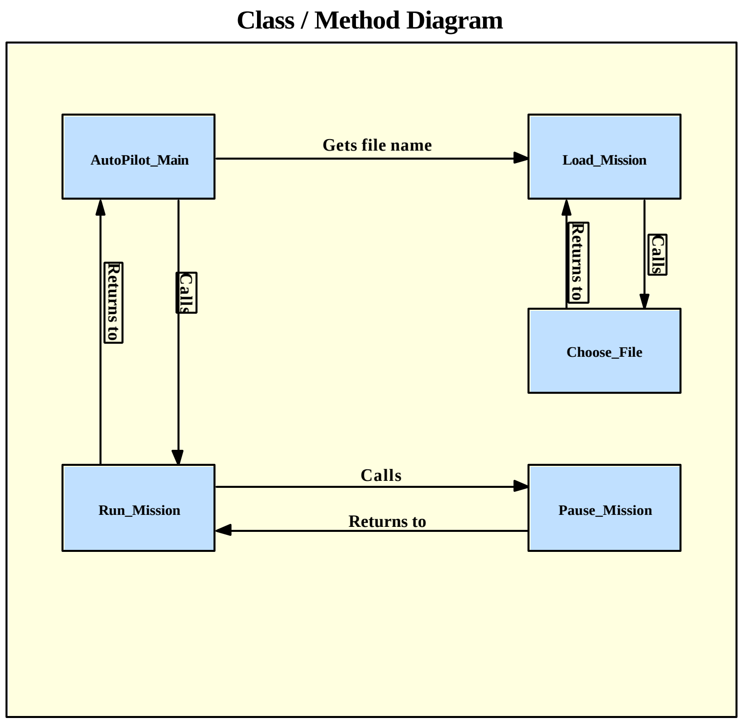


Figure 6. Control flow for the AutoPilot component

2. Mechanistic Design. The mechanistic design consisted of a limited number of detailed use cases and only two sample sequence diagrams. Some of the detailed use cases and sequence diagrams can be seen in Tables 6 through 11 and Figures 7 and 8.

Use Case ID:	1		
Use Case Name:	Load Mission		
Created By:	Matt Behnke, Dan Turnas	Last Updated By:	Matt Behnke, Dan Turnas
Date Created:	2/26/2002	Date Last Updated:	2/26/2002
Actor:	User, Autopilot		
Description:	The Autopilot prompts the user to select a mission file.		
Preconditions:	The Autopilot must be started and is in an idle state.		
Postconditions:	Mission file selected.		
Normal Course of Events:	1. The user selects the Load Mission command. 2. Autopilot displays directory tree. 3. User navigates directory tree. 4. User selects a file. 5. User confirms file to be loaded.		
Alternative Courses:	4. User doesn't find desired file. 5. User cancels Load Mission		
Exceptions:	N/A		
Includes:	N/A		
Assumptions:	N/A		
Notes and Issues:	None		

Table 6. Detailed Load Mission use case

Use Case ID:	2		
Use Case Name:	Run Mission		
Created By:	Matt Behnke, Dan Turnas	Last Updated By:	Matt Behnke, Dan Turnas
Date Created:	2/26/2002	Date Last Updated:	2/28/2002
Actor:	User, Autopilot		
Description:	The Autopilot runs a mission file.		
Preconditions:	The Autopilot must be started, be in an idle state, and the mission file must be loaded.		
Postconditions:	Mission is run.		
Normal Course of Events:	1. User selects Run Mission 2. Mission file parsed correctly. 3. Schedules data to be sent according to the duration specified in the mission file. 4. Sends data packet via use case Send Data Packet		
Alternative Courses:	1. Unable to parse mission file. 2. Alert user that the mission file cannot be parsed. 3. Put Autopilot in an idle state.		
Exceptions:	N/A		
Includes:	Send Data Packet (UCID: 3)		
Assumptions:	N/A		
Notes and Issues:	None		

Table 7. Detailed Run Mission use case

Use Case ID:	3		
Use Case Name:	Send Data Packet		
Created By:	Matt Behnke, Dan Turnas	Last Updated By:	Matt Behnke, Dan Turnas
Date Created:	2/28/2002	Date Last Updated:	2/28/2002
Actor:	Autopilot, 1553 data bus, User		
Description:	Sends information (speed, heading) to the DID through the 1553 bus.		
Preconditions:	Mission is running and data must be scheduled to be sent to the DID.		
Postconditions:	None		
Normal Course of Events:	1. Scheduled data is sent to the DID		
Alternative Courses:	None		
Exceptions:	N/A		
Includes:	N/A		
Assumptions:	N/A		
Notes and Issues:	None		

Table 8. Detailed Send Data Packet use case

Use Case ID:	4		
Use Case Name:	Stop Mission		
Created By:	Matt Behnke, Dan Turnas	Last Updated By:	Matt Behnke, Dan Turnas
Date Created:	2/28/2002	Date Last Updated:	2/28/2002
Actor:	Autopilot, User		
Description:	Autopilot cancels mission		
Preconditions:	Mission must be running		
Postconditions:	Autopilot is idle		
Normal Course of Events:	1. User selects Cancel Mission. 2. Autopilot pauses mission. 3. Autopilot asks for confirmation. 4. User confirms command. 5. Put Autopilot in an idle state.		
Alternative Courses:	4. User doesn't confirm command. 5. Autopilot resumes mission.		
Exceptions:	N/A		
Includes:	Pause Mission (UCID 5)		
Assumptions:	N/A		
Notes and Issues:	None		

Table 9. Detailed Stop Mission use case

Use Case ID:	5		
Use Case Name:	Pause Mission		
Created By:	Matt Behnke, Dan Turnas	Last Updated By:	Matt Behnke, Dan Turnas
Date Created:	2/28/2002	Date Last Updated:	2/28/2002
Actor:	Autopilot, User		
Description:	Autopilot pauses mission		
Preconditions:	Mission must be running		
Postconditions:	Autopilot is paused		
Normal Course of Events:	1. User selects Pause Mission. 2. Autopilot pauses mission.		
Alternative Courses:	None		
Exceptions:	N/A		
Includes:	N/A		
Assumptions:	N/A		
Notes and Issues:	None		

Table 10. Detailed Pause Mission use case

Use Case ID:	6		
Use Case Name:	Resume Mission		
Created By:	Matt Behnke, Dan Turnas	Last Updated By:	Matt Behnke, Dan Turnas
Date Created:	2/28/2002	Date Last Updated:	2/28/2002
Actor:	Autopilot, User		
Description:	Autopilot resumes mission		
Preconditions:	Mission must be paused		
Postconditions:	Autopilot is running mission.		
Normal Course of Events:	1. User selects Resume Mission. 2. Autopilot resumes mission.		
Alternative Courses:	None		
Exceptions:	N/A		
Includes:	N/A		
Assumptions:	N/A		
Notes and Issues:	None		

Table 11. Detailed Resume Mission use case

SEQUENCE DIAGRAMS

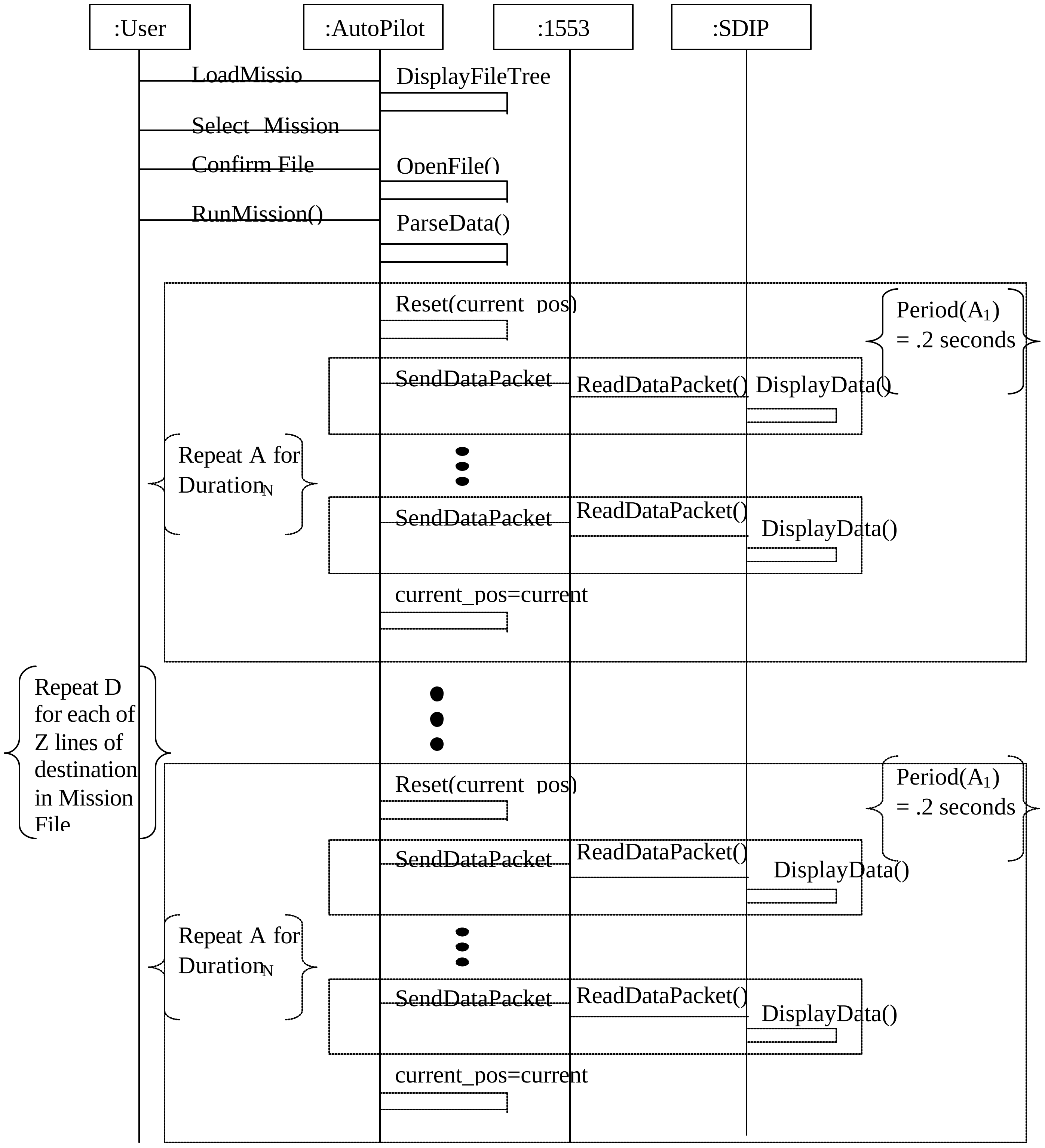


Figure 7. Normal sequence of events

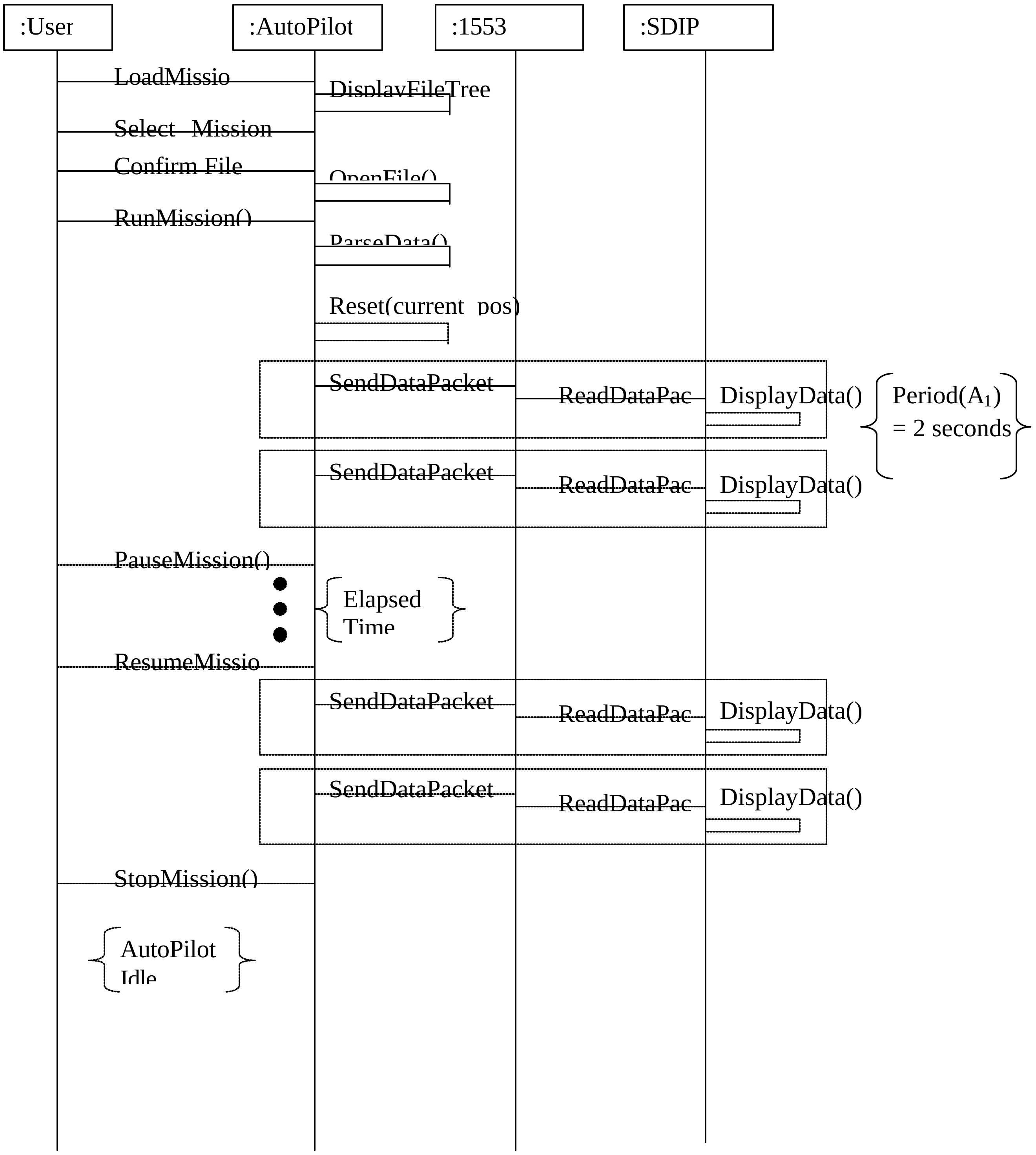


Figure 8. Pause, Resume, and Stop Mission functions

3. Detailed Design. The most design effort was focused on the detailed design of the SDIP system. Complete detailed state diagrams were developed, but only for the AutoPilot component. Pseudocode was not developed for the AutoPilot component. The state diagrams can be seen in Figures 9 through 12.

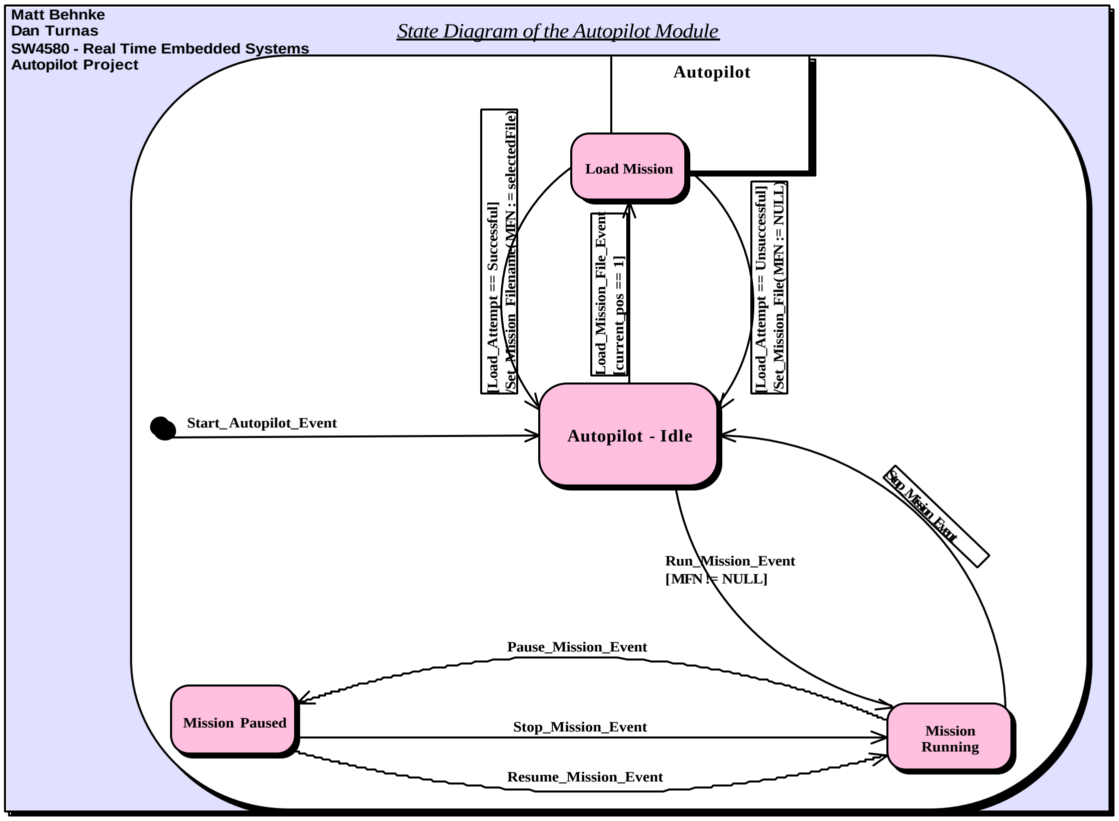


Figure 9. State diagram for the AutoPilot CSC

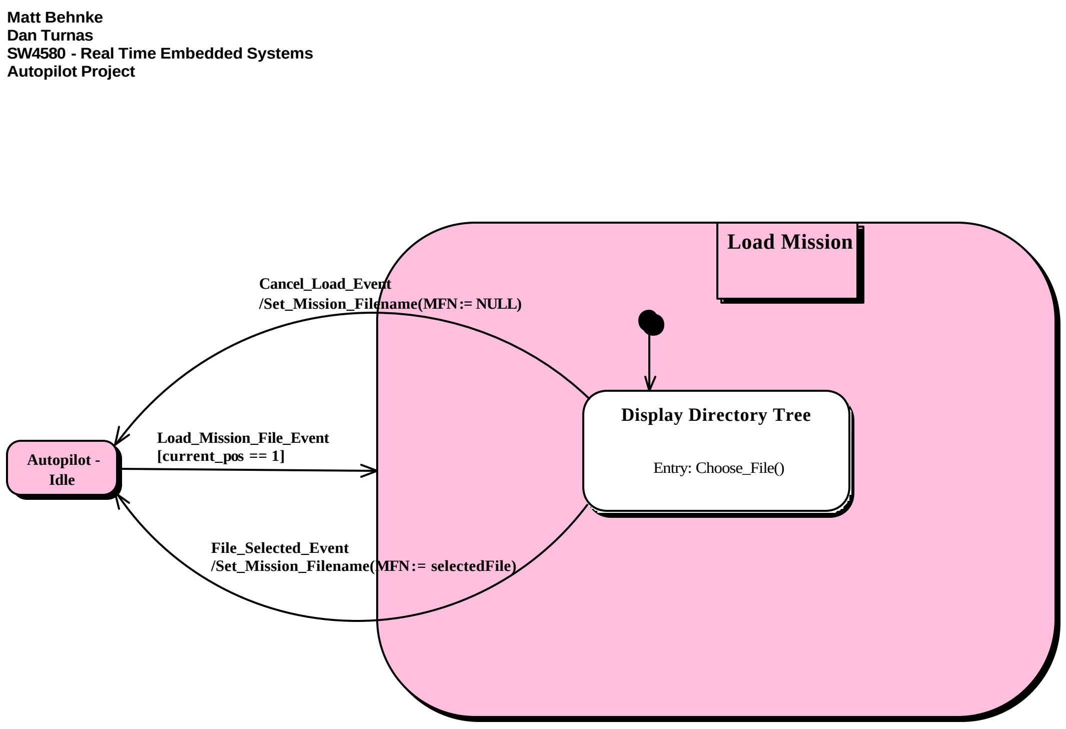


Figure 10. State diagram of the Load Mission function

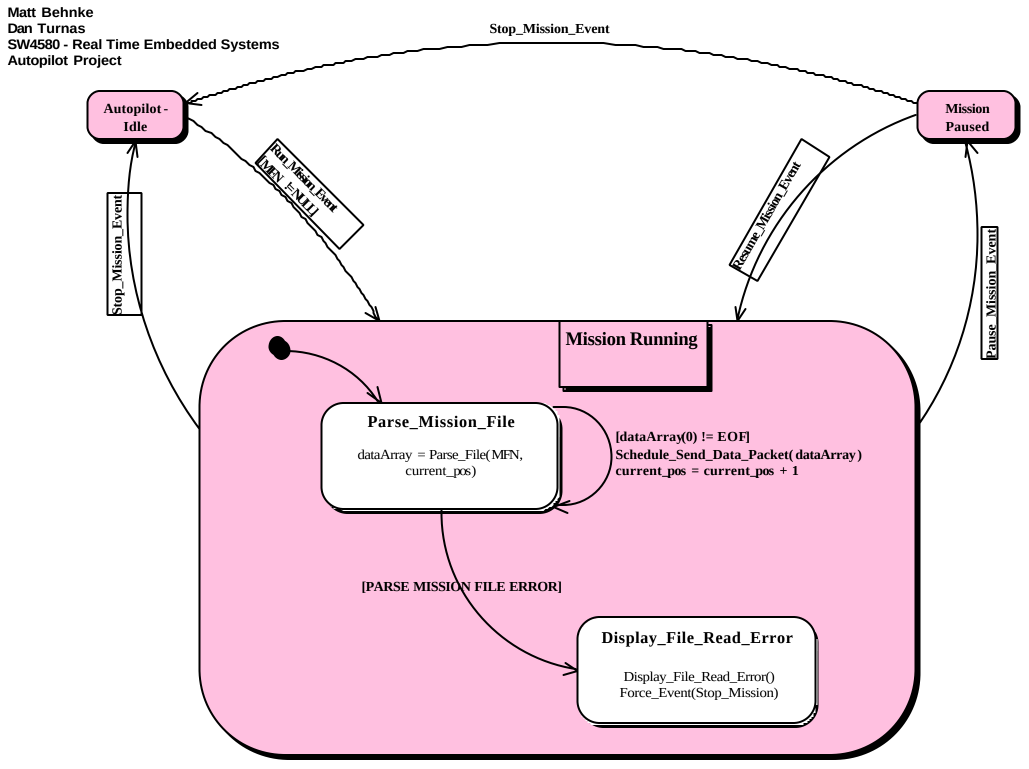
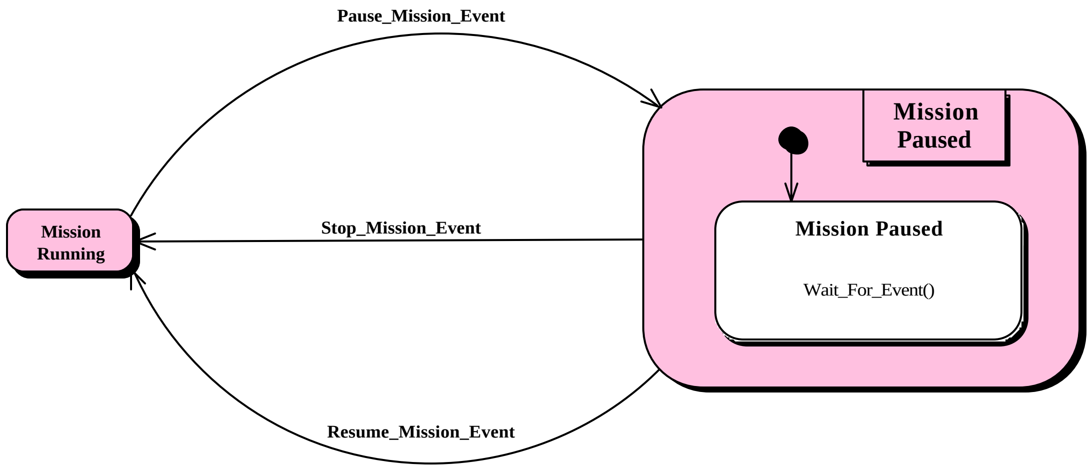


Figure 11. State diagram of the Run Mission function.



Matt Behnke
Dan Turnas
SW4580 - Real Time Embedded Systems
Autopilot Project

Figure 12. State diagram of the Pause Mission function.

c. Testing

The third major activity within the Software Product Engineering KPA is software testing. The SDIP development project did not reach a formal testing phase. Resources were planned and allocated for a thorough testing phase based on Next Gen SW-CMM Level 3 processes. A software test plan was being developed per defined processes when the project was officially cancelled. Upon termination, it was estimated that the test plan was approximately 75% complete. Neither the test coverage outline nor the test cases documentation was developed. These documents were to be created based on completion of the software. Because the first completed software iteration was never delivered to the independent test group, all of the test documentation could not be completed.

The software requirements specification document did establish four verification methods for tracking each requirement through testing. Each of the requirements was analyzed to determine the testing method used to verify that the requirement was correctly implemented in the SDIP system. The four verification methods (as seen in Appendix C) are:

Demonstration: The operation of the system, or a part of the system, that relies on observable functional operation not requiring the use of instrumentation, special test equipment, or subsequent analysis.

Test: The operation of the system, or part of the system, using instrumentation or special test equipment to collect data for later analysis.

Analysis: The processing of accumulated data obtained from other qualification methods. Examples are reduction, interpolation, or extrapolation of test results.

Inspection: The visual examination of system components, documentation, etc.

Each of the 49 requirements is assigned a verification method in the qualification method matrix. A portion of the matrix can be seen in the Table 12. The complete table can be seen in section 6.2 of the Software Requirements Specification Appendix.

Req ID	**Qualification Method**			
	Demonstration	**Test**	**Analysis**	**Inspection**
(020)			X	
(021)	<Requirement Deleted>			
(022)			X	
(023)			X	
(024)	<Requirement Deleted>			
(025)			X	
(026)			X	
(027)			X	
(028)	<Requirement Deleted>			
(029)			X	
(030)			X	
(031)	X			
(032)	X			
(033)	X			
(034)		X		
(035)	X			

Table 12. Partial table of verification methods to be used in testing each requirement

3. Peer Reviews

The third SW-CMM Level 3 KPA, from a developer's point of view, is the Peer Reviews KPA. The Peer Reviews KPA has the fewest number of activities, and was understandably the easiest to achieve. The major peer review activities performed by the SDIP team included: conducting peer reviews according to organizational policies, recording data collected from peer reviews, and having a software quality assurance representative participate in peer reviews. The peer review process defined by Next Gen mandated that all software work products successfully pass a peer review. The SDIP project conducted peer reviews on all of the work products based on the Next Gen process. Metrics were collected for each peer review that was held. Table 13 shows preparation metrics collected during a peer review for the Software Requirements Specification document. Other metrics collected during each peer review included final major and minor faults, peer review duration, and faults per page. Table 13 also shows the attendance of the SQA representative during the peer review.

	Chris Ostrowski (systems engineer)	Karen LaFond (SQA)	Dan Turnas (developer)	John Bohn (developer)	Joe Szafranski (SEPG)
Prep Time (in minutes)	60	30	60	30	45
Major Faults	0	0	0	0	0
Minor Faults	21	2	5	0	8
Questions	3	1	0	0	8

Table 13. Fault metrics collected at June 12th peer review of SRS document

4. Metrics

During the Next Gen SW-CMM Level 3 certification process one of the major opportunity for improvement that was cited was a need for more thorough metrics-based activities. The CMM Level 3 goal of the SDIP development project was to improve the Level 3 processes defining metrics activities. Based on this goal significant effort was allocated for metrics-based activities. Project metrics were defined, collected, analyzed, and used for product and process improvement.

Next Gen allocated resources to form a Process Action Team (PAT) to focus on SDIP metrics. The PAT, along with each of the SDIP developers, used the Goal-Question-Metric process to determine which metrics to collect on the SDIP project. The PAT established six categories of metrics to be collected. These categories were identified by use of the Goal/Question/Metric tables as seen in Appendix B. In Appendix B the full metrics selection tables can be seen. The appendix also shows a complete breakdown of the metrics selected to be collected, analyzed, and reported. Each of the metrics sections is described in detail below. The six metrics categories are as:

Delivery of a project on schedule

Estimation of resource requirements

Management of project within budgeted costs

Product quality

Project communication and collaboration

Size estimation of project work product deliverables

Based on adherence to CMM Level 3 activities, metrics usage became a major part of the SDIP system development effort. A metrics tracking tool was developed in-house to aid the developers in managing the vast amount of metrics that were collected. The Labor Metrics Tracking tool was developed and used. Though complete metrics were sometimes not collected, metrics were collected often enough to give an accurate description of the project status. The metrics were reviewed throughout the entire project lifecycle and can be used by future projects for planning and estimating purposes. A description and analysis of each metrics category can be seen below.

a. Delivery of Project on Schedule

The SDIP development team tracked metrics on percentage of project deliverables that were actually delivered on schedule. The weekly metrics that were collected were based on estimates of percentage completed on each deliverable. As the project was re-planned before a schedule baseline could be established, the final metrics are compared to the re-planned schedule. The variance shown in Table 14 is based on the re-plan schedule and a completion date of July 29, 2002. Based on Table 14, the project was 84% complete with only 44.25% of the scheduled tasks completed. Of the 14 identified project deliverables 5 were completed, 6 were in progress, and 3 were not started. There were no deliverables completed on time.

PROJECT DELIVERABLES	**PLANNED COMPLETION (Original)**	**PLANNED COMPLETION (Re -plan)**	**ACTUAL COMPLETION**	**VARIANCE (DAYS)**
Development Plan	04/02/02	06/24/02	06/27/02	3
Configuration Management Plan	04/02/02	06/13/02	93% completed	N/A
Quality Assurance Plan	04/02/02	05/29/02	06/04/02	4
Requirements Specification	04/17/02	06/17/02	06/18/02	1
Design Documentation	04/19/02	06/12/02	55% completed	N/A
Test Plan	04/19/02	06/13/02	75% completed	N/A
Software Test Coverage Outline	Not scheduled	No date specified	0% completed	N/A
Software Test Cases	Not scheduled	No date specified	0% completed	N/A
Autopilot CSC	05/002	06/24/02	88% completed	N/A
Instrument Panel CSC	Not scheduled	06/24/02	91% complete	N/A
Communications Interface CSC	Not scheduled	06/24/02	71% complete	N/A
1553 Interface CSC	Not scheduled	06/14/02	07/02/02	12
User's Guide, Installation Manual	12/04/02	07/04/02	35% completed	N/A
Post Mortem Report	Not scheduled	07/17/02	09/12/02 est.	40
TOTAL				60

Table 14. Final metrics of schedule events

b. Estimation of Resource Requirements

Metrics were collected on the SDIP development based on initial planning estimates. The metrics were used to determine the accuracy of planning estimates. Each project lifecycle phase was estimated based on the time-boxed project. Metrics were then collected based on hours spent in each lifecycle phase. Table 15 shows the variance in estimated hours versus actual hours spent on each lifecycle phase.

While this portion of metrics collection did help in risk management of the SDIP development, the usefulness of this metrics category will be seen during initial planning on future projects.

LABOR CATEGORIES	**ESTIMATES (HRS)**	**ACTUALS (HRS)**	**VARIANCE (HRS)**
Planning	176	731.50	555.50
Tracking	120	70.00	-50.00
Reviews and Audits	80	228.00	148.00
Analysis	152	130.00	-22.00
Requirements	240	97.00	-143.00
Design/Code (not completed)	1104	836.00	-268.00
Testing (not done)	40	3.00	-37.00
SQA (SQA hours recorded in other categories)	394	0.00	-394.00
SCM (Some SCM hours recorded in other categories)	40	2.00	-38.00
Debriefing	8	244.25	236.25
TOTALS	**2354**	**2341.75**	**-12.25**

Table 15. Final metrics of resources, taken in man hours

c. Management of Project Within Budgeted Costs

The metrics collected in this section were to be used for tracking budgeted costs against costs spent. After the majority of the cost was estimated to be man hours, only work hours were collected for this category. Based on tracking this metric category, it can be seen that each of the developers spent the entire development lifecycle working on the SDIP system. While this conclusion is not remarkable, the metric category was still useful. Future development projects may have costs not limited to man hours.

d. Product Quality

During the metric category identification phase, the product quality category was established. The means to ensure product quality was determined to be regular peer reviews on all lifecycle artifacts. The product quality category tracked metrics associated with peer reviews. Table 16 shows some of the metrics including errors found, major errors, minor errors, meeting duration, and preparation time.

Metric Type	Number of Peer Reviews	Metrics Value (average)
Total Errors	12	40.5
Major Errors	12	0.4
Minor Errors	12	40.1
Preparation Time	7	40.2 minutes
Meeting Duration Time	9	100 minutes

Table 16. Final metrics for product quality

e. Project Communication and Collaboration

The fifth metrics category that was identified was communication and collaboration. This category was collected to monitor meetings, action items, and risks associated with inter-team communication of ideas. Monthly In Process Reviews were held, though only 75% of the time, to brief the customer and management on project status.

Action items were documented from each of the bi-weekly project meetings. Out of the 9 scheduled project meetings, only 4 were actually held. While this metric might indicate poor communication, it is actually misleading. The development team often met and communicated outside of a formal meeting environment. No metrics were collected to reflect this.

f. Estimation of Project Work Product Deliverables

The final metrics category identified was based on estimation of lifecycle artifacts. The estimate for software size was determined at the beginning of the project to be 3.5 KSLOC. The software size at the project termination date was 4.3KSLOC.

This metrics category was also to be used to compare actual lifecycle document size to estimated size. This would then help determine percentage complete for each document. The estimate was never performed at the beginning of the project.

The goal of future metrics collection of this category would be to estimate document size at the beginning of the project lifecycle. Document metrics from the SDIP system will be used to help estimate document sizes from future projects. Table 17 shows the final document metrics for the SDIP system.

MAJOR DELIVERABLE DOCUMENT	# OF PAGES
Software Development Plan (SDP)	24
Software Configuration Management Plan (SCMP)	07 (draft document)
Software Quality Assurance Plan (SQAP)	13
Software Requirements Specification (SRS)	18
Software Design Document (SDD)	10 (Draft document)
Software Test Plan	09 (draft document)

Table 17. Final metrics of work product deliverables

Metrics analysis led to two major project changes. The first change occurred based on information obtained from schedule progress metrics. When it was determined that existing source code was not useable, schedule metrics were referred to. It was decided that schedule metrics indicated that the original project design could not be implemented. A new architecture was implemented. The second major metrics-based decision was to extend the delivery date of the project. At the original deadline, metrics indicated that the project was near completion. A short extension was given to help complete the project.

5. SW-CMM Level 3 Summary

The SDIP development project performed almost all of the SW-CMM Level 3 activities. While some planning and design activities were not performed according to defined processes, all other processes were followed. The development team also improved on several Level 3 opportunities for improvement that were cited during the SW-CMM Level 3 certification process. Requirements management activities were improved, as well as risk management and metrics-based activities. Table 18 shows a sample of how the Level 3 activities within the Integrated Software Management, Software Product Engineering, and Peer Reviews KPAs were implemented on the SDIP development project.

KPA	CF	KP	Key Practice	Document Defined	Project Artifact / Implementation
ISM	Co	1	The project follows a written organizational policy requiring that the software project be planned and managed using the organization's standard software process and related process assets.	* ISM Policy * ISM Standard * ISM Procedure	* SDP
ISM	Ac	10	The project's software risks are identified, assessed, documented, and managed according to a documented procedure.	* ISM Policy * ISM Standard * ISM Procedure * Software Risk Management Procedure * Risk Assessment Worksheet	* Risk matrix in the SDP * Risk section of monthly IPRs * IPR Quad Charts
ISM	Ac	11	Reviews of the software project are periodically performed to determine the actions needed to bring the software project's performance and results in line with the current and projected needs of the business, customer, and end users, as appropriate.	* ISM Policy * ISM Standard * ISM Procedure * In-Process Review Procedure	* Monthly IPRs * Quad Charts
SPE	Co	1	The project follows a written organizational policy for performing the software engineering activities.	* SPE Policy * SPE Standard * SPE Procedure	* Process defined in SDP * SQA Plan * Test Plan
SPE	Ac	2	The software requirements are developed, maintained, documented, and verified by systematically analyzing the allocated requirements according to the project's defined software process.	* SPE Policy * SPE Standard * SPE Procedure * Requirements Management Procedure	* Process defined in SDP * Peer review of requirements * Test Plan
SPE	Ac	3	The software design is developed, maintained, documented, and verified, according to the project's defined software process, to accommodate the software requirements and to form the framework for coding.	* SPE Policy * SPE Standard * SPE Procedure * High Level Design Procedure * Detailed Design Procedure	* SDP * SDD * Peer review of design
SPE	Ac	4	The software code is developed, maintained, documented, and verified, according to the project's defined software process, to implement the software requirements and software design.	* SPE Policy * SPE Standard * SPE Procedure * Code and Unit Test Procedure	* SDP * Peer review of code * SDD * Test Plan
SPE	Me	1	Measurements are made and used to determine the functionality and quality of the software products.		* Test Plan * Metrics collection reports
SPE	Me	2	Measurements are made and used to determine the status of the software product engineering activities.		* IPR Quad charts * Project schedule * Weekly project meetings * Project metrics
PR	Co	1	The project follows a written organizational policy for performing peer reviews.	* PR Policy * PR Standard * PR Procedure	* Peer review records
PR	Me	1	Measurements are made and used determine the status of the peer review activities.	* PR Policy * PR Standard * PR Procedure * PR Templates	* Peer review records on prep time, hours spent, errors found

Table 18. SW-CMM Level 3 SDIP Activities

B. PERFORMANCE TOWARDS SW-CMM LEVEL 4

The second goal of the SDIP project was the execution of SW-CMM Level 4 activities. The activities within the Quantitative Process Management and Software Quality Management KPAs built on experience obtained from SW-CMM Level 3 activities. The Level 3 activities based on metrics collection and analysis are used in Level 4 activities of product and process metrics-based management. Two factors lead to the prevention of performing Level 4 activities.

The first factor was the software maturity level of Next Gen. During the SW-CMM Level 3 certification process, Next Gen identified metrics-based activities as an opportunity for improvement. This improvement would result in metrics experience required to advance towards Level 4 activities. Toward this goal, significant effort was allocated for Level 3 metrics-based activities. Based on the short lifecycle of the SDIP project, resources were not available to both develop Level 3 experience and begin performing Level 4 activities. Therefore, resources were only focused on Level 3 metrics-based activities.

The second factor preventing Level 4 activities was a decision made shortly after the beginning of the project lifecycle. The original goal of process improvement was based on the SW-CMM model. The next logical process improvement goal with this model was certification at Level 4. Shortly after the beginning of the project, a decision was made to transition from SW-CMM Level 4 certification to CMMI Level 3 certification. This decision negated the need for the project team to perform Level 4 activities.

Based on the software maturity level of Next Gen and the decision to move to the CMMI model, SW-CMM Level 4 activities were not performed on the SDIP project.

IV. CONCLUSIONS

Next Gen is dedicated to delivering quality software solutions for major US Army systems. One of the concepts that Next Gen uses to assure the development of quality software is process improvement. Dedication to continuous process improvement allows Next Gen to constantly improve the way software is developed. The use of the Software Engineering Institute's Capability Maturity Model for Software allows Next Gen to measure the level of software development maturity. Recent certification at SW-CMM Level 3 is just one milestone on the journey of process improvement. The next organizational goal is certification at SW-CMM Level 4.

The SDIP was proposed as a pilot project for process improvement within Next Gen. The project was to improve upon SW-CMM Level 3 activities and begin implementing Level 4 processes. The project was developed as a thesis project with business value to the organization. Because the process improvement was performed from the developer's point of view, this thesis focused on activities normally encountered by a software developer. The Level 3 KPAs that were focused on include the Integrated Software Management, Software Product Engineering, and Peer Reviews KPAs. This thesis focused on both of the Level 4 KPAs: Quantitative Process Management and Software Quality Management.

A. PERFORMANCE OF SW-CMM LEVEL 3 ACTIVITIES

1. Integrated Software Management

Estimation was given significant effort by the development team. Estimates were obtained on three major categories: software size, man-hours, and documentation. Software size and man-hour estimates were regularly compared to current project status. The documentation estimates uncovered a need to allocate resources with quality and processes experience to assist in creating the project documents. Post project analysis found that initial estimates were not accurate. This was directly related to the developers' inexperience with determining initial estimates. More experience is necessary in determining more accurate initial estimates. The estimating activities performed by the development team were found to be consistent with SW-CMM Level 3 requirements.

The SDIP development team planning activities were based on a hybrid iterative and incremental lifecycle model. The lifecycle model was composed of planning, designing, coding, and testing phases. Activities within each phase were defined. Due to schedule and budget constraints, some of the lifecycle phases were not realized. The development team did not complete the coding phase and did not encounter a formal testing phase. Due to the extra effort required to establish better metrics-based activities, significant planning resources were required. Planning activities were found to be opportunities for improvement on the SDIP project based on SW-CMM Level 3 requirements.

Risk management activities were performed on the SDIP development project. The risk management on the SDIP development project was broken down into four major categories. The first category was risk assessment. Several risks were identified and ranked. Mitigation techniques were established based on risk severity. The second category was risk tracking. The identification, collection, and analysis of metrics was used to help establish current development status. The status was then used to help monitor potential risks. The third category of risk management was the use of configuration management. CM was planned as a significant component of the development process. The team did not progress far enough into the development lifecycle to require the extensive use of configuration management. The final category of risk management used by the development team was quality assurance. A quality assurance representative was an integral part of the development efforts. The representative assisted in metrics tracking and analysis. The four major risk management activities used on the SDIP project provided significant process improvement based on previous organizational processes.

2. Software Product Engineering

Requirements management was seen throughout the entire development lifecycle. Requirements were collected from the customer and indexed by a unique identification number. Each requirement was assigned to a particular software component to help track the requirement through design and coding. This tracking method would also be used in the testing phase, had the schedule progressed that far. The requirements management activities were found to be consisted with SW-CMM Level 3 requirements.

Software design activities performance can be seen on the SDIP project. Initial use of existing code allowed the development team to use existing design documents. When it was finally determined that existing code could not be used, it was apparent that existing design documents could not be used. By this time, the schedule and resources available did not allow for significant effort to be allocated for formal software design activities. At the time of project cancellation, software coding was nearing completion, even though only a portion of the design was performed. Only an estimated 55% of the design of only one of the three software components was performed. Software design was the major opportunity for improvement area on the SDIP project. SW-CMM Level 3 requirements were not met for software design activities.

Based on the cancellation of the SDIP project, the development team never entered a formal testing phase. Formal test methods were planned for the SDIP system. The verification methods were established for each of the forty-nine identified requirements. Formal test procedures were not yet established at the time of project cancellation. The testing resources had not yet been used to create the formal test procedures. If the schedule would have been extended, there were no factors that would have prevented a thorough application of formal testing. The testing activities performed were consistent with processes defined against SW-CMM Level 3 requirements.

3. Peer Reviews

Next Gen defined processes on peer reviews were followed during the SDIP project. Peer reviews were required on all software work products. Each work product that was developed had a peer review. Detailed metrics were collected on each peer review that was held. A Software Quality Assurance representative attended all peer reviews. The peer review activities were performed in accordance to SW-CMM Level 3 requirements.

4. Metrics

Metrics-based activities were considered a Next Gen Level 3 opportunity for improvement. Considerable resources were allocated for the activities of defining, collecting, and analyzing metrics. A Process Action Team was formed to use the Goal-Question-Metric process to define which metrics were to be collected. A software tool was developed in-house to collect metrics. Metrics were analyzed and used to make several project decisions. Both design and schedule decisions were made based on metrics collected. Once the project was completed, the effectiveness of the metrics collected was assessed. A reduced set of metrics was determined to be most effective. Projects that began after this assessment used the reduced metrics set. Significant process improvements were made on metrics-based Level 3 activities.

B. PERFORMANCE OF SW-CMM LEVEL 4 ACTIVITIES

Two factors prevented Level 4 activities from being performed. The first factor was based on limited project resources. Necessary manpower was not available to both implement Level 3 process improvements and begin performing Level 4 activities within the short duration of the SDIP project. The second factor was a decision made after the beginning of the project to change the process improvement goal from SW-CMM Level 4 certification to the CMMI Level 3 certification. These factors negated the need for Level 4 activities.

C. SUMMARY

The SDIP project was to be developed with the goal of process improvement. SW-CMM Level 3 activities were to be improved upon and Level 4 processes were to be defined. Based on the aggressive project schedule, adequate resources were not available to both improve Level 3 processes and define Level 4 processes. The project used available resources to improve Level 3 processes. Integrated Software Management, Software Product Engineering, and Peer Reviews KPAs were focused on for this project. It was found that significant process improvement was made in risk management, requirements management, and metrics-based activities. Due to effort establishing more improved metrics experience, planning activities were found to be opportunities for improvement. Software Product Engineering and Peer Reviews activities were found to be consistent with previous process levels. The most significant process improvement was seen on metrics-based activities.

The SDIP project goal was software development process improvement. Though Level 4 processes were not established, significant improvements were made on Level 3 processes. The SDIP project was a successful solution towards the goal of continuous process improvement. Future process improvement activities should use the experience gained from the SDIP project.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. BIBLIOGRAPHY

[1] Software Engineering Institute. Software, Systems Engineering, and Integrated Product Development Capability Maturity Models. Spring 2003 <http://www.sei.cmu.edu/cmm/cmms/transition.html>.

[2] Diaz, Michael and King, Jeff. How CMM Impacts Quality, Productivity, Rework, and the Bottom Line. Published in *CrossTalk*, March 2002, pp. 9-14.

[3] Paulk et al. The Capability Maturity Model: Guidelines for Improving the Software Process. New York, NY: Addison-Wesley, 1995.

[4] Olsem, Mike and Schneider, Randy. Achieving SA-CMM Level 2 at PM Abrams. Published in *CrossTalk*, August 2002, pp.8-13.

[5] Fisher et al. Applying the Software Acquisition Capability Maturity Model. Published in *CrossTalk*, August 2002, pp. 4-7.

[6] Fenton, Norma and Pfleeger, Shari Lawrence. Science and Substance: A Challenge to Software Engineers. Published in *IEEE Software*, Vol. 11, No. 4, July 1994

[7] Dymond, Kenneth M. A Guide to the CMM: Understanding the Capability Maturity Model for Software. Annapolis, Maryland: Process Transition International, Inc., 2001.

[8] Brooks, Frederick P. JR. The Mythical Man-Month: Essays on Software Engineering. Reading, MA: Addison-Wesley, 1995.

[9] Leveson, Nancy. Safeware: System Safety and Computers. New York, NY: Addison- Wesley, 1995.

[10] Boehm, Barry W. Software Risk Management: Principles and Practices. Published in *IEEE Software*, Vol. 8, No. 1, January 1991.

[11] Hall, Elaine M. Managing Risk: Methods for Software Systems Development. New York, NY: Addison-Wesley, 1998.

[12] Hatley, Hruschka, and Pirbhai. Process for System Architecture and Requirements Engineering. New York, NY: Dorset House Publishing, 2000.

[13] Leffingwell, Dean. Managing Software Requirements: a Unified Approach. New York, NY: Addison-Wesley, 2000.

[14] Hofmeister, Nord, Soni. Applied Software Architecture. New York, NY: Addison-Wesley, 2000.

[15] Berzins and Luqi. Software Engineering with Abstractions. New York, NY: Addison-Wesley, 1991.

[16] Bruegge, B. and Dutoit, A. (2000). Object-oriented Software Engineering: Conquering Complex and Challenging Systems. Upper Saddle River, NJ: Prentice-Hall, Inc., 2000.

[17] Larman, C. Applying UML and Patterns: An Introduction to Object-Orientated Analysis and Design and the Unified Process (2nd Ed.). Upper Saddle River, NJ: Prentice-Hall, Inc., 2002.

[18] Advanced Logistics Project. Cougaar Open Source Agent Architecture. Winter 2003 <http://www.cougaar.org/index.html>. 16 May 2002.

[19] Binder, Robert V. Testing Object-Oriented Systems. New York, NY: Addison-Wesley, 2000.

[20] Egyed, A., Medvidovic, N., and Gacek, C. Component-Based Perspective on Software Mismatch Detection and Resolution. Published in IEE Software Engineering, Volume 147, Issue 6, December 2000, pp. 225-236.

[21] Gilb, Tom. The 10 Most Powerful Principles for Quality in Software and Software Organizations. Published in *CrossTalk*, November 2002, pp. 4-8.

[22] Daich, Gregory T. Document Diseases and Software Malpractice. Published in *CrossTalk*, November 2002, pp. 23-25.

[23] Weller, Edward F. Using Metrics to Manage Software Projects. Published in *Computer*, Vol. 27, No. 9, September 1994.

[24] Natwick, Gary. Integrated Metrics for CMMI and SW-CMM. Published in *CrossTalk*, May 2003, pp. 4-7.

[25] Pfleeger, Shari Lawrence and McGowan, Clement. Software Metrics in the Process Maturity Framework. Published in *J. Systems and Software*, Vol. 12, 1990.

[26] Pfleeger, S. L. and Fitzgerald, J. C. Jr. Software Metrics Toolkit: Support for Selection, Collection, and Analysis. Published in *Information and Software Technology*, Vol. 33, No. 7, September 1991.

[27] Pfleeger, Fitzgerald, and Rippy. Using Multiple Metrics for Analysis of Improvement. Published in *Software Quality J.*, Vol. 1, 1992.

[28] Perkins, Timothy K. The Nine-Step Metrics Program. Published in *CrossTalk*, February 2001, pp. 16-19.

[29] Myers, Ware. Control the Software Beast with Metrics-Based Management. Published in *CrossTalk*, August 2002, pp. 19-21.

[30] Florence, Al. CMM Level 4 Quantitative Analysis and Defect Prevention. Published in *CrossTalk*, February 2001, pp. 20-23.

[31] Rosenberg, Dr. Linda H. What is Software Quality Assurance?. Published in *CrossTalk*, May 2002, pp. 22-25.

[32] Oman, Paul W. A Case Study in SQA Audits. Published in *Software Quality J.*, Vol. 2, 1993.

[33] Information Technology Office. DASADA Program Site. Winter 2003 <http://www.schafercorp-ballston.com/dasada/index2.html>.

[34] Jones, Capers. Software Cost Estimation in 2002. Published in *CrossTalk*, June 2002, pp. 4-8.

[35] Fulton, Gregory P. SEI CMM Level 5: Lightning Strikes Twice. Published in *CrossTalk*, September 2002, pp. 22-24.

[36] Sheard, Sarah A. Twelve System Engineering Roles. Proceedings of INCOSE Symposium, 1996.

[37] Wells, David. Using DASADA Runtime Probes and Gauges Throughout the Software Lifecycle to Improved Software Quality for the Abrams M1A2 Main Battle Tank. Unpublished Information Paper.

[38] Ray, W. Realizing Adaptive Systems. Proceedings of the OOPSLA 2002 Conference, Seattle, Washington, November 2002.

[39] Pazandak, Paul and Wells, David. ProbeMeister: Distributed Runtime Software Instrumentation. Proceedings of 1st International Workshop on Unanticipated Software Evolution, 2002.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A – INITIAL USER REQUIRMENTS

The initial user requirements list can be seen in this email.

-----Original Message-----
From: Saboe, Michael
Sent: Monday, January 28, 2002 6:01 PM
To: Turnas, Daniel; Gersky, Douglas; Bohn, John
Cc: Pradeep, Kris; Nguyen, Danh; Zobair, Hamza; Bankowski, Elena; Miles, Chris; Slominski, Mark; Menko, Russell
Subject: DID Test Article for Research team -- quote required S:7 Feb 02

Gentlemen (Dan, Doug, John)

We are in need of a test article that we can use for experiments. *I want an estimate and a plan (only for now)* from you three to develop a "Skinny" DID with "cruise control". John Bohn has had discussions with me about this.

This means

- some very small amount of functionality behind a virtual bench fascia.
- Engine speeds, temp, fuel, ground speed, throttle, break, fuel, pressures
- heater, lights,
- way points (lat/long), and
- a "signal to the engine" via a 1553 data packet
- to enable a toy cruise control for "drive to" way point sequence.
- stubs (will send an "inoperable message" or grey out) for all of the other functions only at the top level in the DID -- e.g. diagnostics, fault management,
- This product will be in Ada.
- It will be built by composing the product from elemental routines already existing in the DID.
- all other DID functionality will be left out.
- It will use an Ada 95 compiler, using Ada 83 restrictions
- External Interfaces will use existing DID Ada Specs, (e,g. don't rewrite a 1553 datapacket or 1553 data handler)
- Variable names will be identical and /or naming conventions of the tank will be used
- -- minimize the addition of new variables (the cruise control is the exception)
- estimate the "nice to have" option cost for making this all endian neutral.

At the end of the day,

- I would like a X windows, PC system (this is similar to the OASIS D2T2 system) that I can give to academic and DARPA researchers that is representative of our systems.
- This test article needs to "drop in" to
  - our test environment.
  - a virtual bench environment (D2T2 at the door)
- It needs to be compatible with the statistical usage test models in the future.

You will

- Keep requirements under control
- use our development process, and
- collect metrics,
- do quality control,
- CM, and
- testing as if it were the main battle tank. -- we will use the existing tank test scripts and procedures -- this is very important
- You will have weekly status reports, quad charts and metric reporting
- Demo incremental capability to me, the customer, (and DARPA PIs as required)
- stick to a schedule and budget and track it.
- You will document the interfaces and general structure so that a PI can understand
  - how to interact and
  - install and
  - use the test article in an inexpensive development environment (e.g. free Gnu Gnat Ada compiler)
- You will have to work with our existing lab manuals to document and explain how the "drop - in" interface is used for the PI user.
- Provide transition orientation to NextGen PI staff.

This will be used in many experiments as a calibration tool. Your effort, schedule, defects, etc will be the measure others will try to improve upon.

If we like *your estimate and plan,* and *you can convince us that you three are the ones for establishing the "best we can do",* I will determine a start time and make appropriate arrangements for us to get this done.

It would be nice to have an operational product with some functionality and defined interfaces to the environment two months after start of work.

Mike

APPENDIX B – SOFTWARE DEVELOPMENT PLAN

Skinny Driver's Instrument Panel (SDIP)
Software Development Plan

June 27, 2002

US ARMY TACOM
Tank Automotive Research, Development and Engineering Center
Next Generation Software Engineering
AMSTA-TR-R
Mailstop #265
Warren, MI 48397-5000

	______________________________	__________
	RAUL ONORO, SENIOR CONSULTANT AUTHOR	DATE
APPROVAL	______________________________	__________
	CHRISTOPHER OSTROWSKI SYSTEMS ENGINEER	DATE
APPROVAL	______________________________	__________
	JOHN BOHN PROJECT LEAD	DATE
APPROVAL	______________________________	__________
	KAREN LAFOND SQA MANAGER	DATE
APPROVAL	______________________________	__________
	JOSEPH SZAFRANSKI SEPG LEAD	DATE
FINAL APPROVAL FOR USE	______________________________	__________
	MICHAEL S. SABOE, PH.D ASSOCIATE DIRECTOR NEXT GENERATION SOFTWARE ENGINEERING TECHNOLOGY AREA	DATE

Skinny Driver's Instrument Panel
Software Development Plan

DOCUMENT REVISION HISTORY

Version	Date	Author	Description
1.0	4/08/92	John Bohn	Initial draft of document
1.1	6/11/02	Raul Onoro	Document rework after project re-plan made in May 2002. Document was reformatted and merged into the NextGen SDP (draft) Template.
1.2	6/18/02	Raul Onoro	Document updated with input from Chris Ostrowski on Jun/12/02 and other inputs.
1.3	6/18/02	Raul Onoro	Updates from meeting with Chris Ostrowski.
1.4	6/26/02	Raul Onoro	Updates from Peer Review meeting completed 6/25/02.
2.0	6/27/02	Raul Onoro	Updates from follow-up Peer Review meeting held 6/27/02. This version of document was accepted and will be signed-off by management.

Skinny Driver's Instrument Panel
Software Development Plan

TABLE OF CONTENTS

Skinny Driver's Instrument Panel
Software Development Plan

1 INTRODUCTION

The purpose of this document is to describe the Software Development Plan (SDP) for the Skinny Driver's Instrument Panel (SDIP) being developed within the Next Generation (NexGen) software development organization.

1.1 Project Overview

The goal of the SDIP Project is to produce Abrams M1A2 Main Battle Tank simulation software implementing a functionally reduced version of the graphical user interface of the Driver's Integrated Display (DID), and an Autopilot user/system control interface. The SDIP software is also intended to be made available to non-governmental entities such as the TACOM DARPA DARTS Principal Investigators for research and testing purposes.

The project was originally planned to begin February 2002 and end July 01, 2002.

1.2 Project Deliverables

The SDIP project has the following deliverables:

1. Software Development Plan (SDP)
2. Software Requirements Specification (SRS)
3. Software Design Document (SDD)
4. Software Configuration Management Plan (SCMP)
5. Software Quality Assurance Plan (SQAP)
6. Software Test Plan (STP)
7. Software Test Coverage Outline
8. Software Test Cases
9. Lessons Learned Report
10. SDIP CSC (code)
11. Autopilot CSC (code)
12. 1553 Interface CSC (code)
13. User's Guide and Installation Manual

1.3 SDP Revision and Replan

Whenever there is a significant change to the project, the impact of that change is assessed. Based on this assessment, the changes are negotiated with all affected groups. Any resulting changes are then incorporated into the related software plans or work products.

The Software Development Plan for this project will be updated if there are significant changes to:

- Requirements
- Schedule
- Manpower resources
- Major technology changes

On May 24, 2002 a project re-plan was completed due to major changes in technology assumptions. Specifically, it was determined that SDIP would not be based on Oasis Driver’s Integrated Display (DID) D2T2 software.

1.4 Reference Materials

Document Title	Document Number
Software Requirements Specification for the Driver’s Integrated Display of the Abrams M1A2 Tank.	SRS-AS15420 Revision B, March 15, 2000**.**
Skinny Driver’s Instrument Panel Software Requirements Specification.	Version 2.00, 6/18/02. Next Generation Software Engineering Technology Area.
US System/Segment Design Document. Version SW 2.5.1, Driver’s Station. Volume 3-1 of 5.	SS-US00001 December 1997. General Dynamics Land Systems Division.
Software Design Document for the Driver’s Integrated Display of the Block Improved Abrams Tank (M1A2.)	SDD-SA15420 Revision C, 4 April 1997 General Dynamics Land Systems Division.
Data Packet Specifications Volume 2 – DID.	DP-SA15132 Vol 2, Version 5.0, October 1997. General Dynamics Land Systems Division.
NextGen software policies, standards, and procedure documents.	(Various).

1.5 Definitions and Acronyms

Acronym	**Definition**
AGIL	Adaptable Graphics Interface Layer. A tool for developing graphical user interfaces (e.g., application screens)
API	Application Programmer Interface
CID	Commander's Integrated Display
COTS	Commercial Off the Shelf
CSC	Computer Software Component
DARPA	Defense Advanced Research Projects Agency
DARTS	Dependable Automated Reconfigurable Technology for Software
DCS	DCS Corporation
Defects	Software problems logged as STRs and found after official delivery to the customer
DID	Driver's Integrated Display
Errors	Software problems logged as STRS and found prior to official delivery to the customer
E-Team	NextGen Software Engineering team
GB	Gigabytes
GWT	Government Witnessed Testing
H/TEU	Hull/Turret Electronics Unit (M1A2 US)
H/TMPU	Hull/Turret Mission Processing Unit
IPR	In-Process Review meeting
LRU	Line Replaceable Unit
NextGen	Next Generation Software Engineering Technology Area
NPS	Naval Postgraduate School
M1A2	Abrams Main Battle Tank
MB	Megabytes
POC	Point of Contact
PR	Peer Review meeting
R-Team	NextGen Research and Infrastructure team
SDF	Software Development File
SCCB	Software Configuration Control Board
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SCR	Software Change Request
SDD	Software Design Document
SDF	Software Development File
SDIP	Skinny Driver's Instrument Panel
SDP	Software Development Plan
SEPG	Software Engineering Process Group
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SRS	Software Requirements Specification
STP	Software Test Plan
STR	System Trouble Report
SW	Software
SWE	Software Engineer
TACOM	Tank-automotive and Armaments Command
TARDEC	Tank Automotive Research, Development, and Engineering Center

Acronym	Definition
TDP	Abrams Common Software Library Technical Data Package

2 PROJECT ORGANIZATION

2.1 Process Model

The SDIP project will use the traditional "waterfall" software development model. Only one iteration of software development is planned.

2.2 Organizational Structure

The project is wholly contained within the NextGen organization. The project team is described in Section 2.4.

2.3 Organizational Boundaries and Interfaces

The customer is the NextGen Associate Director, Michael S. Saboe, Ph.D. The project team will work with NextGen's SCM, SQA, Software Testing, and SEPG representatives as listed in Section 2.4.

The Point of Contact (POC) is the NextGen Systems Engineer, Chris Ostrowski.

2.4 Project Responsibilities

The project staff will follow NextGen software policies, standards, and procedures. The project staff will also follow the project-specific plans.

Function	Assigned	Responsibilities
Project Lead	John Bohn	• Maintain project schedule. • Develop SDP (with support from consultant). • SW design and coding. • Review and maintain the Project Risks table. • Participate in Peer Reviews. • Approve Test Plan, Test Coverage outline, and Test Cases. • Develop and present the Lessons Learned report.
Software Engineer	Matt Behnke George Hamilton	• SW design and coding. • Participate in Peer Reviews. • Provide input into the Lessons Learned report.
Software Engineer	Doug Gersky	• Develop SRS. • SW design and coding. • Participate in Peer Reviews. • Provide input into the Lessons Learned report.
Software Engineer	Dan Turnas	• Develop SDD. • SW design and coding. • Participate in Peer Reviews. • Provide input into the Lessons Learned report.

Function	Assigned	Responsibilities
SQA Manager	Karen LaFond	• Provide input to project planning activities. • Develop SQA Plan. • Verify the project team activities are compliant with policies, standards, and procedures. • Facilitate definition of measurable goals for product and process quality. • Monitor that IPRs, Peer Reviews, and project meetings are held as planned. • Monitor that project risks are being reviewed, recorded, and tracked. • Report SQA status and issues to NextGen Associate Director. • Raise non-compliance issues (that cannot be resolved at project level) to Associate Director. • Ensure metrics data is collected and recorded. • Participate in Peer Reviews or assign SQA representative. • Provide input into the Lessons Learned report.
SQA Engineer	Karen LaFond's staff	• Perform reviews and audits per SQAP. • Raise concerns of non-compliance issues to Project Lead and SQA Manager. • Witness system testing activities. • Provide input into the Lessons Learned report.
SEPG Lead	Joseph Szafranski	• Coach project team on NextGen software policies, standards, and procedures. • Audit SQA activities. • Provide input into the Lessons Learned report.
SCM Manager	Russell Menko	• Develop SCM Plan for project. • Perform / manage Software Configuration Management activities. • Provide input into the Lessons Learned report.
Software Test Manager	Nadia Abadir	• Manage testing activities. • Approve Test Plan, Test Coverage outline, and Test Cases. • Provide input into the Lessons Learned report.
Software Test Engineer	Nadia Abadir's staff	• Develops Test Plan, Test Coverage Outline, and Test Cases. • Perform Testing.
Systems Engineer	Chris Ostrowski	• Point of Contact for the project. • Assist and coach the project team in project management and technical activities. • Assist in resolving non-compliance issues that cannot be resolved at the project level. • Approve SDP, SQAP, and SCMP documents. • Approve Test Plan, Test Coverage outline, and Test Cases. • Participate in Peer Reviews. • Provide input into the Lessons Learned report.

Function	Assigned	Responsibilities
Associate Director	Michael S. Saboe, Ph.D.	• NextGen Associate Director. • Customer. • Resolve non-compliance issues that cannot be resolved at the project level. • Provide input into the Lessons Learned report. • Provide final approval of the SDP.

3 MANAGERIAL PROCESS

3.1 Management Objectives and Priorities

The goal of the SDIP Project is to produce Abrams M1A2 Main Battle Tank simulation software implementing a functionally reduced version of the graphical user interface of the Driver's Integrated Display (DID), and an Autopilot user/system control interface.

The SDIP software is also intended to be made available to non-governmental entities such as the TACOM DARPA DARTS Principal Investigators for research and testing purposes.

3.2 Assumptions, Dependencies and Constraints

Assumptions:

- The software engineering team is available full time.
- Ada 95 programming language will be used.
- PC based Redhat Linux (version 7.3) operating system will be used.

Dependencies:

- People are time constrained with other projects and commitments.

Constraints:

- The project has been scheduled in a "time box" beginning February 2002 and ending July 01, 2002.

3.3 Project Tracking Methods

The Project will be tracked using the following methods:

- **Project Team Meetings** – In these meetings, the current progress is compared against the schedule. Issues and concerns are discussed and addressed. New Action Items are reviewed and assigned. Project Meeting minutes will be published within three business days from the meeting date.
- **Monthly IPRs** – IPRs are conducted each month with NextGen senior management. At the IPR, the Project Lead reports on the status of project progress, open action items, deviations from the software development plan, project risks, and recommendations for remedial action. IPRs are the primary means of reporting problems to senior management.
- **Action Items** – Action Items for the project will be collected, logged, and tracked to completion. The Project Lead manages the Action Items log for the project and reports on their status.

3.4 Metrics

Metrics data are being collected throughout the project. The main purpose of these metrics is to manage the project, gather information about the project, to define software development baseline(s), and to assist future process improvement activities.

The metrics data identified for this project address the following areas:

1. Delivery of projects on schedule.
2. Estimation of resource requirements.
3. Management of project within budgeted cost.
4. Product Quality.
5. Project communication and collaboration.
6. Estimation of project work product deliverables.

Refer to Appendix A for details on the metrics selected for this project.

In order to maintain progress on the project, the tracking metrics in the following table will be used. These metrics identify situations requiring immediate attention and specify the actions to be taken to resolve the situations.

Tracking Metric	**Lower Control Limit**	**Corrective Action – Exceeded Lower Control Limit**	**Upper Control Limit**	**Corrective Action – Exceeded Upper Control Limit**
Project Schedule: • Number of tasks and milestones actually completed by date measured. **Metric 1-2.**	70% of tasks and milestones actually completed on or ahead of schedule.	• Project Lead addresses issues affecting schedule. • Project Lead re-plans the project.	130% of tasks and milestones actually completed on or ahead of schedule.	Project Lead adjusts schedule for earlier completion.
Management Reviews: • Number of IPR meetings held (per plan). **Metric 5-2.**	75% of planned IPR meetings actually held.	• Project Lead coordinates IPR meeting with NextGen Associate Director.	N/A	N/A
Team Communication: • Number of project team meeting held (per plan). **Metric 5-9.**	80% of planned meetings actually held.	• Project Lead resolves issues preventing meetings.	N/A	N/A
Process: • Number of Peer Reviews held (per plan). **Metric 5-19**.	85% of work products completing Peer Reviews per plan.	• Project Lead resolves issues preventing Peer Reviews. • Project Lead ensures Peer Reviews are being held.	N/A	N/A

3.5 Risk Management

The initial set of project risks and their assessment are identified in the table below.

Risk	**Severity**	**Probability**	**Risk Level**	**Mitigation Strategy**
Diversion of development team to other projects including GWT.	High	High	High	• Keep management and customer up to date on status.
Inability to meet project goals due to inexperienced development team.	High	High	High	• Make extensive use of NextGen and DCS domain experts and NPS course work.
Inability to meet project goals due to development and implementation of CMM related processes.	High	High	High	• Ensure project team gets "on the job" training of the new CMM related processes. • Get assistance from senior consultants.
Aggressive schedule.	High	Medium	High	• Develop detailed project plan. • Ensure resources are available to work on the project. • Address schedule delays early. • Work with customer in the prioritization of functionality to be delivered.
Previously performed class work may not be sufficient to meet the SDIP project specific requirements.	High	Medium	High	• Rework class project to meet SDIP project requirements.
Development environment is flawed or not understood.	High	Medium	Medium	• Work with domain experts.
Domain experts are unavailable.	High	Medium	Medium	• Develop contacts with other personnel with required expertise.
Misunderstood requirements.	High	Medium	Medium	• Work closely with customer.
PC unavailable.	High	Medium	Medium	• Acquire PC for project as soon as possible.
Requirements creep.	Medium	Low	Low	• Requirements management.
Hardware failure / data loss.	High	Low	Low	• Backups.
Illness and vacations.	Medium	Low	Low	• Early identification of vacation plans.

The SDIP project Risk Management table is located in the SDIP Software Development Folder (SDF). The project risks will be evaluated and the table updated on a periodic basis. High level risks will be reviewed at SDIP IPR meetings.

3.6 Staffing Plan

The team will be composed of NextGen E-Team members who have gained some experience with the tank software performing maintenance as required by STRs or testing. The Project lead and Software Engineers are expected to be available at a minimum 65% of their time during the duration of the project. See Section 2.4 for Roles, Assignments, and Responsibilities.

4 DEVELOPMENT PROCESS

4.1 Methods, Tools, and Techniques

4.1.1 Computing Environment

The Project requires the following computing environment:

- PC equipment with the following minimum configuration:
  - Pentium III 450 Mhz processor
  - 128 MB of RAM
  - 10 GB of Disk
  - Display capable of supporting 1024 x 768 resolution
  - Ethernet interface card
- PC based Readhat Linux (version 7.3) operating system
- Ada 95 compiler (GNAT)
- Ada 95 debugger
- X Windows client software (X Manager)
- AGIL and related libraries and scripts
- Microsoft Office 2000
- Microsoft Project 2000
- Clearcase (SCM tool)
- 1553 interface card (optional)

The Project requires the follow Lab environment:

- M1A2 System Bench for testing.
- Bench 1553 Emulator software for PC.

4.1.2 Software Development Folder

The Project will maintain a Software Development Folder (SDF) where all project related documents and other materials will be maintained. The electronic project folder is located on the **"Ice"** NT server under the **"\\Ice\Nt-shared\Sdip"** directory path.

At a minimum, the SDF contains the project deliverables listed in Section 1.2.

4.2 Software Documentation

The Project's staff will produce various documents. See Section 1.2 for the list of Project Deliverables. See Section 6. for details.

4.3 Project Support Functions

The SDIP Project is supported by personnel from the SCM, SQA, Test, and SEPG organizations.

4.3.1 Software Trouble Reports

Software problems discovered after the start of formal testing by the Software Test organization will be logged formally as Software Trouble Reports (STRs). STRs will be reviewed by a project specific SCCB and SCRs will be created as appropriate.

Significant problems or updates to project documents, discovered after their Peer Review and management sign-off, will be logged formally as STRs.

Software problems logged as STRS and found prior to official delivery to the customer are named "**errors**".

Software problems logged as STRs and found after official delivery to the customer are named "**defects**".

STRs and SCRs will be stored in the SCM database.

4.4 Post Delivery Support

The NextGen R-Team will be responsible for providing software support for the SDIP system after SDIP has been released for use.

4.5 Process Tailoring

This project will follow the NextGen software development policies, standards, and procedures.

Project tailoring (deviation from NextGen process), is described in the table below.

Document Title	Document Number	Tailoring Performed
None identified		

5 PROJECT ESTIMATES AND CRITICAL RESOURCES

5.1 Work Elements

The project will consist of the activities identified in the master project schedule located in the SDIP Software Development Folder (SDF).

5.2 Dependencies

The project has the following dependencies:

- Autopilot software based on previous class work.

5.3 Resource Estimates

The project requires four software engineers for the core work and a systems engineer performing technical consulting and acting as the POC.

Software engineers are assumed to be available 65%, which allows for sick time, 1 week trip to NPS, and E-Team activity support.

The original estimates for the amount of software to be developed for SDIP are shown below.

SDIP Computer Software Components	KLOC
SDIP CSC	1.5
Autopilot CSC	0.5
1553 Interface CSC	1.5
Total	**3.5**

The initial effort estimates below are based on SDIP code size estimate of 3.5 KLOC.

Development Phase/Activity	Effort Hrs. Estimate
Planning	176
Tracking	120
Reviews and Audits	80
Debriefing	8
Analysis	152
Requirements	240
Design/Code	1104
Testing	40
SQA	394
SCM	40
Total	2354

5.4 Budget and Resource Allocation

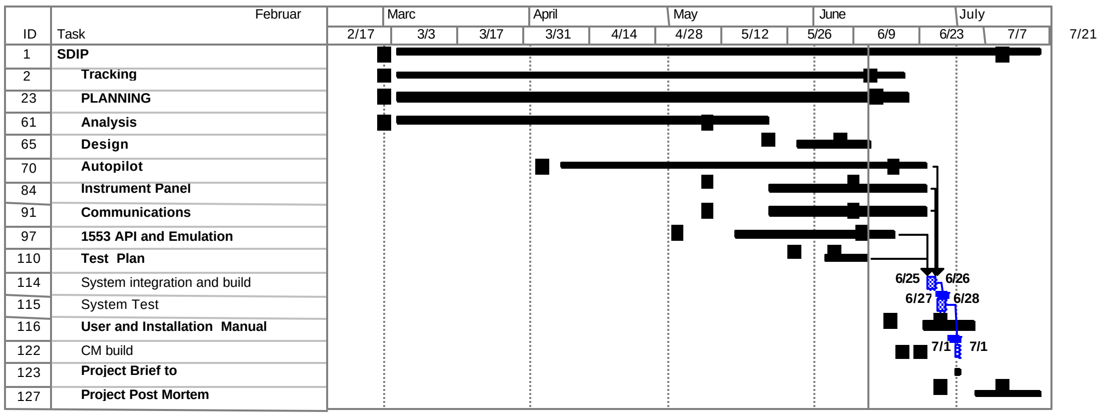
The SDIP Project is not managing a detailed budget. Therefore, a budget was not allocated to the separate phases of the project. Instead, cumulative project costs are determined by the labor metrics collected by the project.

Task assignments and resource allocations are defined and tracked in the project schedule. For details see the project schedule located in the SDIP Software Development Folder (SDF).

5.5 Schedule

The schedule will be maintained separately in a Microsoft Project 2000 file located in the SDIP Software Development Folder (SDF).

Below is a high level view of the initial schedule.



ID	Task
1	**SDIP**
2	**Tracking**
23	**PLANNING**
61	**Analysis**
65	**Design**
70	**Autopilot**
84	**Instrument Panel**
91	**Communications**
97	**1553 API and Emulation**
110	**Test Plan**
114	System integration and build
115	System Test
116	**User and Installation Manual**
122	CM build
123	**Project Brief to**
127	**Project Post Mortem**

5.6 Project Training Plan

The project requires the following training:

- Two day AGIL training course supplied by DCS. This training was held May 16 and 17, 2002. Training session was attended by: John Bohn, Doug Gersky, Dan Turnas, and Chris Ostrowski.

6 SUPPORT FUNCTIONS

6.1 Software Configuration Management

The Software Configuration Management organization plans, advises, and manages (under configuration management) the project's work products and deliverables. A Software Configuration Management (SCM) Plan describes in detail the SCM activities. The SCM Plan is located in the SDIP Software Development Folder.

All project deliverables will be placed under configuration control after completing their Peer Review and management sign-off. Refer to Section 1.2 for the list of project deliverables.

Once the SDIP CSCI is ready for testing, the Project Lead will notify the SCM manager. SCM will then perform the software "build" for the SDIP CSCI using scripts or instructions from the development team. The SCM software "build" is then turned over to the Software Test organization for testing.

6.2 Software Quality Assurance

The SQA organization plans, advises, reviews, and insures the project is following the NextGen policies, standards, and procedures.

The SQA activities for this project include:

- Planning
- Tracking
- Reviewing and Auditing (product and process)
- Project Debriefing / Post Mortem Analysis

For further details, see the SQA Plan located in the SDIP Software Development Folder.

6.3 Software Testing

The Software Testing organization plans, develops, executes, and documents results of software tests.

The Software Test Plan, Test Coverage Outline, and Test Cases describe in detail the software testing activities for this project. For further details, see the software test documents located in the SDIP Software Development Folder.

APPENDIX A – GOAL/QUESTION/METRICS

NOTATIONS:

- Table cells that are highlighted represent metric collection activities at the end of the project.
- "**Collected by & When**" column choices: Now, At a given Document Review, Monthly IPR, or Post-Development.
- "**Projected Effort**" column scale: Low (< 10 minutes); Medium (11-30 minutes); High (> 30 minutes) to collect.

Goal 1: Improve the Ability to Deliver Projects on Schedule

Question	#	Metric	Collected by & When	Projected Effort	To be Reported By & When
Are projects being completed on time?	1-1	Number of tasks and milestones planned for completion by date measured	SQA Engineer & Project Lead – Monthly for IPR	Low	SQA Manager for IPR
	1-2	Number of tasks and milestones actually completed by date measured	SQA Engineer & Project Lead – Monthly for IPR	Low	SQA Manager for IPR, Project Lead – post mortem
	1-3	Project duration variance	Project Lead – Post development	Medium	Project Lead – post mortem
Are lifecycle phases being completed on schedule?	1-4	Percent of lifecycle phases completed on time	Project Lead – Post development	High	Project Lead – post mortem
	1-5	Phase duration variance	Project Lead – Post development	High	Project Lead – post mortem
	1-6	Percent phase milestones/tasks completed on time	Project Lead – Post development	High	Project Lead – post mortem
Are deliverables being completed on schedule?	1-7	Percent project deliverables completed on schedule	Project Lead – Post development	Medium	Project Lead – post mortem
	1-8	Delivery variance	Project Lead – Post development	Medium	Project Lead – post mortem
	1-9	Average project deliverable variance	Project Lead – Post development	Medium	Project Lead – post mortem
Are senior management reviews conducted in a timely manner?	1-9	Percent senior management reviews (IPRs) conducted per schedule	SQA Engineer – Post development	Medium	SQA Manager – post mortem
Are SQA activities scheduled?	1-10	Percent of SQA activities scheduled	SQA Engineer – Post development	Medium	SQA Manager – post mortem
Are SQA activities	1-11	Percent SQA activities	SQA Engineer –	Medium	SQA Manager –

Question	#	Metric	Collected by & When	Projected Effort	To be Reported By & When
completed on time?		completed on time	Post development		post mortem
Are project requirements managed?	1-12	Total number of requirements	Project Lead - Monthly for IPR	Low	Project Lead for IPR, Project Lead – post mortem
	1-13	Number of requirements added	Project Lead - Monthly for IPR	Low	Project Lead for IPR, Project Lead – post mortem
	1-14	Number of requirements deleted	Project Lead - Monthly for IPR	Low	Project Lead for IPR, Project Lead – post mortem
	1-15	Number of existing requirements that were modified	Project Lead - Monthly for IPR	Low	Project Lead for IPR, Project Lead – post mortem

Goal 2: Improve Ability to Estimate Resource Requirements

Question	#	Metric	Collected by & When	Projected Effort	To be Reported By & When
Is effort estimated for the project?	2-1	Project has effort estimates (Y/N)	Project Lead @ SDP	Low * # People	Project Lead @ 1st IPR after SDP has Peer Review
Do project effort estimates accurately predict actual project effort?	2-2	Project effort variance	Project Lead @ Post development	Medium	Project Lead @ Post Development
Is effort estimated for each lifecycle phase?	2-3	Percent of lifecycle phases with effort estimates	Project Lead @ SDP	Low * # People	Project Lead @ 1st IPR after SDP has Peer Review
Do lifecycle phase efforts estimates accurately predict actual lifecycle phase effort?	2-4	Phase effort variance	Project Lead @ Post development	Medium	Project Lead @ Post Development
Is effort estimated for each phase deliverable? SRS & SDD	2-5	Percent of deliverables for which there are estimates	Project Manager @ SDP	Low	Project Lead @ 1st IPR after SDP has Peer Review
Do deliverable effort estimates accurately predict actual deliverable effort? SRS & SDD	2-6	Variance per deliverable for which there are estimates	Project Lead @ Post development	Medium	Project Lead @ Post Development
Is risk identification being performed?	2-7	Projects risks been identified (Y/N)	Project Lead @ SDP	Low	Project Lead @ 1st IPR after SDP has Peer Review
Were there any unexpected risks?	2-8	How many per project?	Project Lead @ Post development	Medium	Project Lead @ Post Development

Goal 3: Manage Project Within Budgeted Cost

Question	#	Metric	Collected by & When	Projected Effort	To be Reported By & When
Have all cost categories been estimated for this project?	3-1	Does the project have costs estimated by appropriate categories? (Y/N) **Category:** Labor hours. (Collected by activity, e.g., Planning, Tracking, Design/Code, ...) Using average labor rate per team member roles. SDIP project will NOT estimate or track costs for other categories such as: • Materials • Training • Subcontractors • Hardware • Software – COTS • Travel	Project Lead @ SDP	Medium	Project Manager @ SDP
Do project cost estimates accurately predict actual project costs by category?	3-2	Project cost variance overall	Project Lead @ Post Development	Medium	Project Lead @ Post Development
	3-3	Percent of project cost variance overall	Project Lead @ Post Development	Medium	Project Lead @ Post Development
	3-4	Cost variance per category.	Project Lead @ IPR	Medium	Project Lead @ IPR

Goal 4: Improve Delivered Quality

Question	#	Metric	Collected by & When	Projected Effort	To be Reported By & When
How effective is the process in detecting defects?	4-1	Number of Project defects detected per deliverable after the deliverable has been signed off. (STRs)	SQA Engineer – As occurs	Low	SQA Manager @ Monthly SQA IPR
How early are errors identified?	4-2	Types of errors detected per phase	SQA Engineer – As occurs	Low	SQA Manager @ Monthly IPR
	4-3	Average Peer Review preparation time.	SQA Engineer – As occurs	Medium	SQA Manager @ Monthly IPR
	4-4	Average number of errors found at Peer Reviews	SQA Engineer – As occurs	Low	SQA Manager @ Monthly IPR
	4-5	Average number of major errors found at Peer Reviews	SQA Engineer – As occurs	Low	SQA Manager @ Monthly IPR
	4-6	Average number of minor errors found at Peer Reviews	SQA Engineer – As occurs	Low	SQA Manager @ Monthly IPR
How effective are we in resolving defects?	4-7	Average time to resolve project defects (STRs)	SQA Engineer – As occurs	Low	SQA Manager @ Monthly IPR
How satisfied are our customers?	4-8	Customer satisfaction survey rating	SQA Engineer – Post development	Medium	SQA Manager @ Monthly IPR

Goal 5: Improve Project Communication and Collaboration

Question	#	Metric	Collected by & When	Projected Effort	To be Reported By & When
Are senior management meetings (IPRs) planned and conducted?	5-1	Number of IPRs planned	SQA Engineer – 1 per month	Low	SQA Manager @ Monthly IPR
	5-2	Number of IPRs held	SQA Engineer – monthly	Low	SQA Manager @ Monthly IPR
	5-3	Percent of IPRs for which materials to be reviewed were distributed on-time (within 1 business day) prior to the meeting	SQA Engineer – monthly	Low	SQA Manager @ Monthly IPR
	5-4	Number of IPR meetings that started on-time – within 5 minutes of posted meeting notice.	SQA Engineer – monthly	Low	SQA Manager @ Monthly IPR
	5-5	Number of IPR meetings where action items addressed	SQA Engineer – monthly	Low	SQA Manager @ Monthly IPR
	5-6	Total number of SDIP Action Items per month	NextGen Dept. Admin. – monthly	Low	SQA Manager @ Monthly IPR
	5-7	Total number of SDIP CLOSED Action Items per month	NextGen Dept. Admin. – monthly	Low	SQA Manager @ Monthly IPR
Are project meetings planned and conducted?	5-8	Number of project meetings planned	Project Lead – as scheduled	Low * # projects	SQA Manager @ Monthly IPR
	5-9	Number of project meetings held	Project Lead – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-10	Percent of project meetings for which materials to be reviewed were distributed on-time (within 2 business days) prior to the meeting	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-11	Number of project meetings with an agenda	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-12	Number of project meetings which addressed at least 80% of the agenda items	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-13	Number of project meetings for which minutes were published on-time (within 2 business days)	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-14	Number of project meetings that started on-time (within 5 minutes of posted meeting notice)	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR

Question	#	Metric	Collected by & When	Projected Effort	To be Reported By & When
	5-15	Number of project meetings rescheduled due to lack of quorum	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-16	Total number of Action Items (all project meetings) per month	Project Lead – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-17	Total number of CLOSED Action Items (all project meetings) per month	Project Lead – as they occur	Low * # projects	SQA Manager @ Monthly IPR
Are Peer Reviews (PRs) planned and conducted?	5-18	Number of PRs planned	Project Lead – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-19	Number of PRs held	Project Lead – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-20	Percent of PRs for which materials to be reviewed were distributed on-time (within 3 business days) prior to the meeting	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-21	Number of PRs for which minutes were published on-time (within 3 business days)	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-22	Number of PRs that started on-time (within 5 minutes of posted meeting notice)	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-23	Number of PRs rescheduled due to lack of quorum	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-24	Planned length of PR (minutes)	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-25	Actual length of PR (minutes)	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-26	Average Peer Review preparation time (minutes)	SQA Engineer – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-27	Total number of Action Items per month	Scribe – as they occur	Low * # projects	SQA Manager @ Monthly IPR
	5-28	Total number of CLOSED Action Items per month	SQA Engineer – as they occur	Large	SQA Manager @ Monthly IPR
Are project team members adequately trained to conduct meetings?	5-29	Percent of staff who have had training or demonstrated ability to conduct meetings	SEPG Manager – monthly	Medium	SEPG Manager @ Monthly SEPG IPR

Goal 6: Improve Ability to Estimate Project Deliverables

Question	#	Metric	Collected by & When	Projected Effort	To be Reported By & When
What are the sizes of deliverables?	6-1	Estimated KSLOC at start of project By category: New, re-used, Modified	Project Lead @ SDP	Medium	Project Manager @ 1st Monthly IPR after SDP Peer Review
	6-2	Actual KSLOC at end of project by category: New, re-used, Modified	SCM Manager @ Post Development	Medium	SCM Manager @ 1st Monthly IPR after project completion
	6-3	# pages – SDP	SCM Manager @ Post Development	Low	SCM Manager @ 1st Monthly IPR after project completion
	6-4	# pages – SCMP	SCM Manager @ Post Development	Low	SCM Manager @ 1st Monthly IPR after project completion
	6-5	# pages – SQAP	SCM Manager @ Post Development	Low	SCM Manager @ 1st Monthly IPR after project completion
	6-6	# pages – SRS	SCM Manager @ Post Development	Low	SCM Manager @ 1st Monthly IPR after project completion
	6-7	# pages – SDD	SCM Manager @ Post Development	Low	SCM Manager @ 1st Monthly IPR after project completion
	6-8	# pages – STP	SCM Manager @ Post Development	Low	SCM Manager @ 1st Monthly IPR after project completion
	6-9	# pages – STR	SCM Manager @ Post Development	Low	SCM Manager @ 1st Monthly IPR after project completion

APPENDIX C – SOFTWARE REQUIREMENTS SPECIFICATION

Skinny Driver's Instrument Panel (SDIP)
Software Requirements Specification

June 18, 2002

U.S. Army TACOM
Tank Automotive Research, Development, and Engineering Center
Next Generation Software Engineering Technology Area
AMSTA-TR-R
Mailstop #265
Warren, MI 48397-5000

	Douglas J. Gersky **Software Engineer** **Author**	**Date**
Approval	**Christopher Ostrowski** **Systems Engineer**	**Date**
Approval	**John M. Bohn** **Project Lead**	**Date**
Approval	**Karen A. LaFond** **SQA Representative**	**Date**
Final Approval For Use	**Michael S. Saboe, Ph.D.** **Associate Director** **Next Generation Software Engineering Technology Area**	**Date**

Revision History

Version	Date	Author	Description
0.01 Draft	03/06/2002	Douglas J. Gersky	Initial layout of document and initial content. Not reviewed by project team.
0.02 Draft	04/10/2002	Douglas J. Gersky	Rework of initial draft, based on agreed upon document formats and requirements discussions. Reviewed by project team prior to Peer Review.
0.03 Draft	04/15/2002	Douglas J. Gersky	Modified with changes from Pre-Peer review of 0.02 Draft.
1.00	04/23/2002	Douglas J. Gersky	Modified document with changes from Peer Review of 04/18/2002. 1st version submitted to CM. Baseline SRS.
1.10	06/05/2002	Douglas J. Gersky	Major document revision based upon re-plan of SDIP project. Requirements have been renumbered from previous revision. This revision is the baseline document for the re-planned SDIP project.
1.20	06/17/2002	Douglas J. Gersky	Modified document with changes from Peer Review of 06/10/2002.
2.00	06/18/2002	Douglas J. Gersky	Final Peer Review modifications. Version submited to SQA for final approval.

Table of Contents

<This Page Intentionally Left Blank.>

1 Scope

1.1 Identification

This document specifies the engineering and qualification requirements for the Skinny Driver’s Instrument Panel (SDIP) computer software configuration item (CSCI). It will be used as a basis for detailed design and testing.

1.2 SDIP Overview

The SDIP Project’s goal is to produce M1A2 simulation software, implementing a functionally reduced version of the Driver’s Integrated Display (DID) from that vehicle. This modified DID is referred to as the Skinny Driver’s Instrument Panel (SDIP). The software resulting from the SDIP Project is collectively known as the SDIP CSCI. The SDIP CSCI is intended to be made available to non-government entities for research purposes. The focus of this research will be the development and addition of software monitoring probes to U.S. Army weapons systems.

1.3 Reference Materials

Document Title	Document Number
Skinny Driver’s Instrument Panel Software Development Plan	SDIP-22-4-1, Version 2.0, <Date TBD> Next Generation Software Engineering Technology Area. TARDEC
US System/Segment Design Document Version SW 2.5.1, Driver’s Station Volume 3-1 of 5	SS-US00001 December 1997 General Dynamics Land Systems Division
Software Design Document for the Driver’s Integrated Display of the Block Improved Abrams Tank (M1A2)	SDD-SA15420 Revision C, 4 April 1997 General Dynamics Land Systems Division
Data Packet Specifications Volume 2 – DID	DP-SA15132 Vol 2, Version 5.0, October 1997 General Dynamics Land Systems Division

1.4 Requirement Identification & Document Notation

Within this document, all software requirements are contained in statements that use the verb ‘shall’. The notation **(###)** prefaces all such requirement statements and is utilized to aid requirement traceability. All software requirements identified in this manner shall be met.

Within this document, references to document sections are denoted by the notation **[#]**.

1.5 Acronyms, Terms, and Definitions

Acronym, Term	Definition
ACSL	Abrams Common Software Library
Ada	High level programming language.
AGIL	Adaptable Graphical Interface Library
API	Application Program Interface
Autopilot CSC	A user interface CSC of the SDIP CSCI.
CDU	Commander's Display Unit
CID	Commander's Integrated Display
COTS	Commercial Off The Shelf
CSC	Computer Software Component. A logical collection of CSUs associated with a specific set of related software functionalities.
CSCI	Computer Software Configuration Item. A logical collection of CSCs.
CSU	Computer Software Unit. An element specified in the design of a CSC that is separately testable.
DECU	Driver's Electronic Control Unit
DID	Driver's Integrated Display
GCDP	Gunner's Control and Display Panel
HEU	Hull Electronics Unit
M1A1/2 & M1A2 SEP	U.S. Army 'Abrams' Main Battle Tank variants.
MBT	Main Battle Tank
M/OSB	Menu Option Select Button
MPU	Mission Processing Unit
NDI	Non-Developmental Item
NextGen	Next Generation Software Engineering Technology Area
SDIP	Skinny Driver's Instrument Panel
SDIP CSC	A user interface CSC of the SDIP CSCI.
SDIP CSCI	Refers to all constituent CSCs of the SDIP project.
SDIP Project	The combined work effort involved in creating the SDIP CSCI.
SEP	System Enhancement Package
TACOM	Tank-Automotive & Armaments Command
TARDEC	Tank Automotive Research, Development, and Engineering Center
TEU	Turret Electronics Unit
UI	User Interface

2 Functional Requirements

2.1 SDIP CSCI Operating Environment

(046) The SDIP CSCI shall be developed using the Ada programming language. **(001)** The SDIP CSCI shall be developed to run in the Linux operating system. The SDIP project will use the Adaptable Graphical Interface Library (AGIL) to develop the user interfaces required by the SDIP CSCI.

2.2 1553 API and Emulation CSC

(048) The SDIP CSCI shall be developed using the 1553 API and Emulation CSC, an NDI. This NDI offers the ability to transmit/receive 1553 data packets across both a TCP/IP network and a 1553 Data Bus. **(045)** The SDIP CSCI shall be capable of using either TCP/IP or the 1553 Data Bus communication protocol for interprocess communication.

2.3 SDIP CSCI Processes

The SDIP CSCI is intended to implement a stand-alone, PC based M1A2 simulation centered around a virtual DID LRU graphical user interface [4.1] and an Autopilot user/system control interface [4.2]. The virtual DID resulting from the requirements contained herein is referred to as the SDIP CSC. **(002)** The other M1A2 display simulations, the CID and GCDP, shall not be available as part of the SDIP CSCI. The Autopilot defined by the requirements herein is a user interface which acts as the system control interface of the SDIP CSCI and is referred to as the Autopilot CSC. The Autopilot CSC and the SDIP CSC run as independent processes when the SDIP CSCI is invoked.

2.4 Processing of 1553 Data Packets

(004) The SDIP CSCI shall maintain the ability to process 1553 data packets in a manner consistent with the actual method used within a M1A2 tank. That is, the SDIP and Autopilot CSCs maintain the M1A2 1553 packet specifications for interprocess communication of data.

2.4.1 SDIP CSC

(005) All M1A2 1553 Data Packets transmitted and received by the M1A2 DID shall be transmitted and received by the SDIP [3]. **(044)** The SDIP shall have the ability to receive M1A2 1553 Data Packets not normally received by the M1A2 DID. **(006)** The SDIP shall emulate a subset of M1A2 DID user interface visual reactions/updates to received 1553 data packets. This subset of reactions/updates contains the following:

- Update velocity.
- Update compass heading.
- Switch to/from "Steer To" screen.
- Update engine rpm.
- Change M/OSB conditional states (off/on, low/high, etc.).
- Fuel Level

2.4.2 Autopilot CSC

(008) The Autopilot shall be capable of transmitting and receiving M1A2 1553 data packets. **(009)** The Autopilot shall be capable of transmitting all M1A2 1553 data packets required to be received by the SDIP. **(011)** The Autopilot shall be capable of receiving M1A2 1553 data packets from the SDIP.

2.5 Creation of 1553 Data Packets

2.5.1 Autopilot 1553 Data Packet Creation Methods

(013) The Autopilot shall create M1A2 1553 data packets through two methods: direct and file.

2.5.1.1 Autopilot File

(014) The Autopilot file data creation method shall be started through user interaction with menus/widgets of the Autopilot user interface [4.2]. The Autopilot file data creation method uses a file that is hereafter referred to as the Mission File. **(015)** The Autopilot shall support its own proprietary Mission File format. **(016)** The Mission File shall contain all data necessary to construct valid M1A2 1553 data packets of the types processed by the entire SDIP CSCI [3]. **(017)** The Mission File format shall also be used for the data capture capability of the Autopilot [4.2].

2.5.1.2 Autopilot Direct 1553 Data Packet Creation

(019) The Autopilot shall allow direct M1A2 1553 data packet creation through manipulation of menus/widgets, etc. contained on its user interface [4.2].

2.5.2 SDIP Direct 1553 Data Packet Creation

(047) The SDIP shall allow direct M1A2 1553 data packet creation through manipulation of menus/widgets contained on its user interface [4.2].

3 Data Requirements

3.1 SDIP CSCI M1A2 1553 Data Packet Categories

(020) The SDIP CSCI shall implement two categories of M1A2 1553 data packets: Project Critical and System Status. The purpose of System Status and Project Critical packets is to maintain and augment the complete 1553 data bus schedule, as specified in the document "Data Packet Specifications Volume 2 – DID". Project Critical packets are intended to be monitored for content. **(022)** Project Critical packets shall be populated with data that emulates values, as would normally be seen in the corresponding data packet as it exists on an operational M1A2 1553 data bus. System Status packets are not intended to be monitored for content. **(023)** System Status packets shall not be required to be populated with data values that emulate data, as would normally be seen in the corresponding data packet, as it exists on an operational M1A2 1553 data bus.

3.2 Autopilot Transmit / SDIP Receive Functionality

The Autopilot CSC is the process which transmits M1A2 1553 data packets to the SDIP CSC. Both Project Critical and System Status packets are transmitted by the Autopilot CSC [2.4.1].

3.2.1 Project Critical Group A

This group of packets contains those that are normally transmitted to the M1A2 DID LRU. In the SDIP CSCI the SDIP CSC simulates the DID, the destination of these packets in a M1A2 system. In the SDIP CSCI the Autopilot CSC simulates the TEU, the source of these data packets in a M1A2 system. **(025)** The following group of Project Critical data packets shall be implemented:

Data Packet ID	**Data Packet Description**	**Source LRU**	**Destination LRU**
DP0400.2_DEV_PWR_ST	Device Power Status	TEU	DID
DP0600.2_AUTO_ST	Automotive Status	TEU	DID
DP0800.1_NAV_HEADING	Pos/Nav Heading	TEU	DID
DP0900.2_LOW_RATE_NAV_OUTPUT	Pos/Nav Low Rate Data	TEU	DID
DP1800.2_WAYPT_DATA	Waypoint Data	TEU	DID

3.2.2 Project Critical Group B

This group of packets contains those that are not transmitted to the M1A2 DID LRU. In the SDIP CSCI the SDIP CSC simulates the TEU, the destination of these packets in a M1A2 system. In the SDIP CSCI the Autopilot CSC simulates the HEU and DECU, the source of these data packets in a M1A2 system. **(026)** The following group of Project Critical data packets shall be implemented:

Data Packet ID	**Data Packet Description**	**Source LRU**	**Destination LRU**
DP2401.DECU_CTL	DECU Control	HEU	TEU
DP2501.DECU_ST	DECU Status	DECU	TEU

3.2.3 System Status

In the SDIP CSCI the SDIP CSC simulates the DID, the destination of these packets in a M1A2 system. In the SDIP CSCI the Autopilot CSC simulates the TEU, the source of these data packets in a M1A2 system. **(027)** The following group of System Status data packets shall be implemented:

Data Packet ID	Data Packet Description	Source LRU	Destination LRU
DP1102_DIAGNOSTIC_ST	Diagnostic Status	TEU	DID
DP1502_MODE_CTRL	Mode Control	TEU	DID
DP2601.2_CB_ST	Circuit Breaker Status	TEU	DID
DP2802_C-W-FLT_SYM	C W System Fault Summary	TEU	DID
DP2902_BIT_CMD	BIT Command	TEU	DID

3.3 SDIP Transmit / Autopilot Receive Functionality

The SDIP CSC is the process which transmits M1A2 1553 data packets to the Autopilot CSC. Both Project Critical and System Status packets are transmitted by the SDIP CSC [2.4.2].

3.3.1 Project Critical

In the SDIP CSCI the SDIP CSC simulates the DID, the source of these data packets in a M1A2 system. In the SDIP CSCI the Autopilot CSC simulates the TEU, the destination of these packets in a M1A2 system. **(029)** The following group of Project Critical data packets shall be implemented:

Data Packet ID	Data Packet Description	Source LRU	Destination LRU
DP0300.1_DEV_PWR_CTL	Device Control	DID	TEU
DP1600.1_BK_NAV_UPDATE	Pos/Nav Control	DID	TEU
DP1900.1_WAYPT_REQ-ST	Waypoint Request Status	DID	TEU

3.3.2 System Status

In the SDIP CSCI the SDIP CSC simulates the DID, the source of these data packets in a M1A2 system. In the SDIP CSCI the Autopilot CSC simulates the TEU, the destination of these packets in a M1A2 system. **(030)** The following group of System Status data packets shall be implemented:

Data Packet ID	Data Packet Description	Source LRU	Destination LRU
DP1002_DIAGNOSTIC_CTL	Diagnostic Control	DID	TEU
DP1202.1_NH_TRIM_CAL	NH Trim Calibration	DID	TEU
DP1702_MODE_REQ-ST	Mode Request Status	DID	TEU
DP2301.1_CB_CTL	Circuit Breaker Control	DID	TEU
DP3002_ST_DATA	ST Data Element	DID	TEU

4 User Interface Requirements

4.1 SDIP CSC

The SDIP CSC user interface models the M1A2 DID display and other user interface components of the DID LRU contained in that vehicle. **(031)** The SDIP CSC user interface shall be based upon the DID user interface as diagrammed in the document "System/Segment Design Document, volume 3-1 of 5, Driver's Station". **(032)** The SDIP CSC user interface shall recreate that portion of the DID menu structure that is normally invoked from user interactions with the DID display. **(033)** The SDIP CSC user interface shall recreate the waypoint "Steer To" display, which in the M1A2 system is invoked by actions performed on the CID. **(034)** The SDIP CSC interface shall invoke its waypoint display based on actions performed on the Autopilot CSC user interface [4.2].

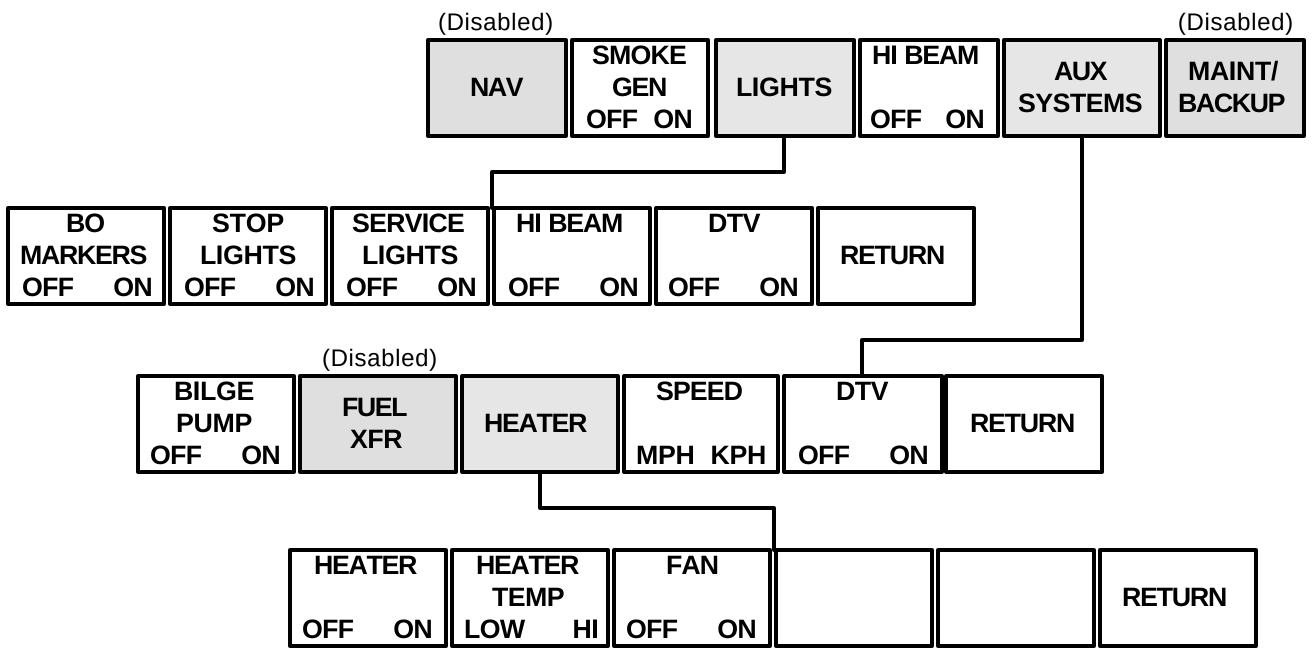
(049) User selection of an enabled M/OSBs on the SDIP user interface shall cause an icon to be displayed on the SDIP user interface, if selecting the corresponding button on an M1A2 DID causes an icon to be displayed on its interface. **(035)** Only a subset of the M/OSBs available on the M1A2 DID shall be enabled on the SDIP user interface. This subset contains some of those M/OSBs whose state is communicated in the Project Critical M1A2 1553 data packets transmitted and received by the SDIP CSC. **(036)** The SDIP M/OSBs that shall remain enabled are listed below and shown in their DID menu positions in the accompanying figure.

- Main Menu (Combat Mode)
  - Smoke Gen Off/On
  - Lights
  - Hi Beam Off/On
  - Aux Systems

- Lights (Main Menu M/OSB)
  - BO Markers Off/On
  - Stop Lights Off/On
  - Service Lights Off/On
  - Hi Beam Off/On
  - DTV Off/On
  - Return

- Aux Systems (Main Menu M/OSB)
  - Bilge Pump Off/On
  - Speed Mph/Kph
  - DTV Off/On
  - Return

- Heater (Aux Systems Menu M/OSB)
  - Heater Off/On
  - Heater Temp Low/Hi
  - Fan Off/On
  - Return



4.2 Autopilot CSC

The Autopilot is the user interface which acts as the system control interface of the SDIP CSCI. **(037)** The Autopilot shall be implemented in a window separate from the SDIP window. **(038)** The Autopilot shall have two modes of operation; Direct and Mission. Direct Mode allows real-time adjustment of velocity and course related data, which in turn is transmitted to the SDIP [3.2]. Mission Mode allows the use of a Mission file [2.5.1.1] for data transmission to the SDIP [3.2]. **(039)** Mission Mode functions shall not be available while the Autopilot is in Direct mode. **(040)** Direct Mode functions shall not be available while the Autopilot is in Mission Mode. **(041)** The Autopilot UI shall include controls which allow users to perform the following functions:

Mode:	**Function:**	**Description:**
Direct	Set Waypoint	Perform steps to set a waypoint for use by the SDIP.
Direct	Steer To Waypoint	Communicate to SDIP to 'steer to' a specific waypoint.
Direct	Adjust Velocity	Increase/Decrease the velocity displayed on the SDIP.
Direct	Adjust Heading	Allow 0° to 360° adjustment displayed on the SDIP.
Direct	Record Mission	Record all Direct Mode adjustments to a Mission file.
Direct	Clear Waypoint	Remove a previously set waypoint.
Mission	Load Mission	Open an existing Mission file.
Mission	Run Mission	Process an open Mission file.
Mission	Pause Mission	Suspend processing of an open Mission file.
Mission	Resume Mission	Continue processing an open Mission file.
Mission	Stop Mission	Stop processing of an open Mission file.
Any	Exit SDIP CSCI	Terminate the processes of the SDIP CSCI.

5 Security Requirements

The SDIP CSCI, as released to principal investigators, includes partial specifications of the M1A2 interprocess communication protocol. That is, a subset of the 1553 data bus, data packet format is included as part of the SDIP CSCI. **(042)** The SDIP CSCI shall therefore operate under the same security requirements as imposed by the M1A2 software system. **(043)** All of the SDIP CSCI's documentation, code, and technical specifications are considered 'Limited Dissemination' material and shall not be released unless directed by the Associate Director of NEXTGEN Software, AMSTA-TR-R, Warren MI 48379-5000 or a higher authority. The 1553 data bus, data packet specification content of the SDIP CSCI is further covered by the Arms Export Control Act (22 USC 2761 et seq) or executive order 12470. Export is therefore restricted.

6 Qualification Provisions

6.1 General

This section specifies the method(s) to be used to ensure each requirement of Section 3 has been satisfied. Qualification methods include:

a) Demonstration: The operation of the system, or a part of the system, that relies on observable functional operation not requiring the use of instrumentation, special test equipment, or subsequent analysis.

b) Test: The operation of the system, or part of the system, using instrumentation or special test equipment to collect data for later analysis.

c) Analysis: The processing of accumulated data obtained from other qualification methods. Examples are reduction, interpolation, or extrapolation of test results.

d) Inspection: The visual examination of system components, documentation, etc.

A Verification Matrix, which relates software requirements to the specific qualification method(s) to be used, is shown in the following section. Shaded boxes in the matrix indicate which test method will be used to verify the software requirement.

6.2 SDIP Software Requirements Verification Matrix

Sec #	Req ID	Qualification Method			
		Demonstration	Test	Analysis	Inspection
2.1	(001)	■			
2.3	(002)				■
	(003)	<Requirement Deleted>			
2.4	(004)		■		
2.4.1	(005)		■		
2.4.1	(006)	■			
	(007)	<Requirement Deleted>			
2.4.2	(008)		■		
2.4.2	(009)		■		
	(010)	<Requirement Deleted>			
2.4.2	(011)		■		
	(012)	<Requirement Deleted>			
2.5.1	(013)	■			
2.5.1.1	(014)	■			
2.5.1.1	(015)				■
2.5.1.1	(016)	■			
2.5.1.1	(017)	■			
	(018)	<Requirement Deleted>			
2.5.1.2	(019)	■			

Sec #	Req ID	Qualification Method			
		Demonstration	Test	Analysis	Inspection
3.1	(020)			■	
	(021)	<Requireme nt Deleted>			
3.1	(022)			■	
3.1	(023)			■	
3.2	(024)	<Requirement Deleted>			
3.2.1	(025)			■	
3.2.2	(026)			■	
3.2.3	(027)			■	
	(028)	<Requirement Deleted>			
3.3.1	(029)			■	
3.3.2	(030)			■	
4.1	(031)	■			
4.1	(032)	■			
4.1	(033)	■			
4.1	(034)		■		
4.1	(035)	■			
4.1	(036)	■			
4.2	(037)	■			
4.2	(038)	■			
4.2	(039)	■			
4.2	(040)	■			
4.2	(041)	■			
5	(042)				■
5	(043)				■
2.4.1	(044)			■	
2.2	(045)	■			
2.1	(046)				■
2.5.2	(047)			■	
2.2	(048)			■	
4.1	(049)	■			

Appendix A Software Requirements / Document Section Mapping

Software Requirement	Document Section# / Requirement Description	
(001)	2.1	SDIP System developed to run under Linux OS.
(002)	2.3	CDI & GCDP displays are not part of SDIP CSCI.
(003)		<Requirement Deleted>
(004)	2.4	Process 1553 data bus packets consistent with actual method used in M1A2 system.
(005)	2.4.1	SDIP transmits & receives all 1553 data packets received by M1A2 DID.
(006)	2.4.1	SDIP emulates a subset of M1A2 DID updates/reactions to 1553 data packets.
(007)		<Requirement Deleted>
(008)	2.4.2	Autopilot transmits & receives 1553 data packets.
(009)	2.4.2	Autopilot transmits all 1553 data packets required by SDIP.
(010)		<Requirement Deleted>
(011)	2.4.2	Autopilot receives 1553 data packets from SDIP.
(012)		<Requirement Deleted>
(013)	2.5.1	Autopilot creates M1A2 1553 data packets through two methods: direct and file.
(014)	2.5.1.1	Data 1553 data packets created through interaction with Autopilot UI.
(015)	2.5.1.1	Autopilot supports its own proprietary Mission File format.
(016)	2.5.1.1	Mission file contains all data necessary to construct 1553 data packet.
(017)	2.5.1.1	The mission file used for the data capture function of the Autopilot.
(018)		<Requirement Deleted>
(019)	2.5.1.2	Autopilot allows direct M1A2 1553 data packet creation through manipulation of menus/widgets contained on its user interface
(020)	3.1	Project Critical & System Status packet types.

Software Requirement	**Document Section#**	**/ Requirement Description**
(021)		<Requirement Deleted>
(022)	3.1	Project Critical packets are populated with valid simulated data.
(023)	3.1	System Status packets are not required to be populated with valid simulated data.
(024)		<Requirement Deleted>
(025)	3.2.1	Project Critical packets transmitted by Autopilot – DID.
(026)	3.2.2	Project Critical packets transmitted by Autopilot – TEU.
(027)	3.2.3	System Status packets transmitted by Autopilot – DID.
(028)		<Requirement Deleted>
(029)	3.3.1	Project Critical packets transmitted by SDIP – TEU.
(030)	3.3.2	System Status packets transmitted by SDIP – TEU.
(031)	4.1	SDIP UI based on DID UI as specified in DID S/SDD.
(032)	4.1	SDIP recreates a portion of DID menu structure triggered from user interactions with DID.
(033)	4.1	SDIP user interface recreates waypoint "Steer To" display.
(034)	4.1	SDIP waypoint functionality triggered by interaction with Autopilot.
(035)	4.1	Subset of the M/OSBs available on the DID shall be enabled on SDIP.
(036)	4.1	Enabled SDIP M/OSB list.
(037)	4.2	Autopilot implemented in separate window.
(038)	4.2	Autopilot implements two modes of operation.
(039)	4.2	Autopilot Mission mode functions not available in Direct mode.
(040)	4.2	Autopilot Direct mode functions not available in Mission mode.
(041)	4.2	Controls contained on Autopilot UI.

Software Requirement	**Document Section# / Requirement Description**	
(042)	5	SDIP inherits M1A2 system security requirements.
(043)	5	SDIP System’s documentation, code, and technical specifications considered ‘Limited Dissemination’ material.
(044)	2.4.1	SDIP CSC can receive M1A2 1553 Data Packets not normally received by the M1A2 DID.
(045)	2.2	SDIP CSCI capable of using either TCP/IP or the 1553 Data Bus communication protocol for interprocess communication.
(046)	2.1	SDIP CSCI developed using the Ada programming language.
(047)	2.5.2	SDIP allows direct M1A2 1553 data packet creation through manipulation of menus/widgets contained on its user interface
(048)	2.2	SDIP CSCI development will include the use if the 1553 API and Emulation CSC, an NDI.
(049)	4.1	Selection of enabled M/OSBs on the SDIP user interface will display an icon on the SDIP user interface, if selecting the corresponding button on an M1A2 DID displays an icon on its interface.

APPENDIX D – SOFTWARE CONFIGURATION MANAGEMENT PLAN

Skinny Driver's Instrument Panel Software Configuration Management Plan

July 12, 2002

US ARMY TACOM
Tank Automotive Research, Development and Engineering Center
Next Generation Software Engineering Technology Area
AMSTA-TR-R
Mailstop #265
WARREN MI 48397-5000

	RUSSELL H. MENKO SCM MANAGER AUTHOR	DATE
APPROVAL	CHRISTOPHER OSTROWSKI SYSTEMS ENGINEER	DATE
APPROVAL	JOHN BOHN PROJECT LEAD	DATE
APPROVAL	KAREN LAFOND SQA MANAGER	DATE
FINAL APPROVAL FOR USE:	MICHAEL S. SABOE, PH.D. ASSOCIATE DIRECTOR NEXT GENERATION SOFTWARE ENGINEERING TECHNOLOGY AREA	DATE

Revision History

Version	Date	Author	Description
0.1	12 March 2002	R. Menko	Initial Draft
0.2	26 April 2002	R. Menko	Update from 1st Peer Review
0.3	19 June 2002	R. Menko	Rewrite to address re-plan
0.4	10 July 2002	R. Menko J. Turner	Incorporate comments from 2nd Peer Review

Skinny Driver's Instrument Panel
Software Configuration Management Plan

Table of Contents

1. Introduction

The purpose of this Software Configuration Management (SCM) Plan (SCMP) is to describe the configuration management procedures followed for the Skinny Driver's Instrument Panel (SDIP).

2. Scope

This plan applies to all software and Next Generation Software Engineering Technology Area (NextGen) developed or modified work products (e.g. software documentation, code, plans, test artifacts, processes) within the NextGen organization, as related to the SDIP program. It shall comply with standard NextGen SCM processes and procedures, and shall be in force unless the Associate Director of the Next Generation Software Technology Area approves a written Request for Deviation (RFD). It shall also apply to software received from external sources, for configuration control purposes.

3. References

Document	Number	Version	Date
Skinny Driver's Instrument Panel Software Development Plan	SDIP-22-4-1	2.0	June 27, 2002
Skinny Driver's Instrument Panel Software Quality Assurance Plan	SDIP-25-4-1	1.0	June 4, 2002

4. Acronyms and Definitions

Acronyms	**Definitions**
ClearCase	Configuration Control software, from Rational Software Corporation
Deliverable Software	Software delivered to the customer as called for by the SDP
NextGen	Next Generation Software Engineering Technology Area
NGSEL	Next Generation Software Engineering Laboratory
RFD	Request For Deviation
SCCB	Software Configuration Control Board
SCM	Software Configuration Management
SCMDB	SCM Data Base
SCMP	SCM Plan
SCR	Software Change Request
SDF	Software Development Folder
SDIP	Skinny Driver's Instrument Panel
SEPG	Software Engineering Process Group
SQA	Software Quality Assurance
STR	System Trouble Report
Support Software	Non-deliverable software required for building or testing the application

5. Roles and Responsibilities

The following table identifies the roles and responsibilities of Project Staff with respect to SCM:

Role	Responsibilities
Project Lead	• Assign work as appropriate • Review all System Trouble Reports (STRs) • Provide STR initial/change information to SCM, as changes occur. • Chair the Software Configuration Control Board (SCCB).
Software Developers	• Submit source code and related documentation to SCM control, as applicable • Create and maintain SCM build scripts
Systems Engineer	• Provide input during SCCB Meetings • Review and approve SCMP
Software Test Manager	• Participate in the SCCB • Submit all test documentation, including Plan, Procedures, and Results, to SCM for placing under configuration control
Software Test Engineer	• Participate in the SCCB • Record all testing results, for placing under configuration control
SCM Manager	• Develop SCM Plan • Participate in project reviews • Ensure the SCM plan, processes, and standards are followed • Participate in SQA audits of SCM activities
SCM Engineer	• Assist the Chair of the SCCB as required • Place all submitted items under configuration control • Perform all SCM builds in compliance with the schedule posted in the SDP
Software Quality Assurance (SQA) Manager	• Participate in the SCCB • Perform audits and reviews of SCM-related activities and material
SQA Engineer	• Perform audits and reviews of SCM-related activities and material

6. Configuration Control

All deliverable and support software and related documentation used for the SDIP will be placed under configuration control. This includes peer review documentation, test documentation and scripts, and meeting minutes and related documents.

SDIP items will be placed under initial configuration control following successful completion of a peer review and, for documentation, following completion of the signature cycle. Third party software will be placed under configuration control upon receipt.

7. Naming Conventions

Standard NextGen Ada naming conventions shall be followed. Executables shall be named as generated during the build process as defined in scripts provided to SCM by Software Development personnel.

System releases shall be named using the prefix "SD" to indicate the Skinny DIP program. E.g. the first release will be called SD1.0.

8. Repositories

SCM shall store all source code and related documentation in Rational ClearCase repositories, residing on the artemis2 Sun server, located in the NextGen engineering lab.

9. Access Control

Only SCM personnel shall access files within the SCM ClearCase repository. Source code files shall be checked in or out by direction of the SCCB. All other files, such as test procedures and peer reviews, shall be checked in/out by SCM at the request of the User.

10. Build Management

SCM shall perform builds per direction of the SCCB. These builds shall be performed using build scripts provided and maintained by Development personnel and shall be built on the Linux platform defined in the SDIP Software Development Plan. Upon completion of a build, the executable code shall be delivered to the Project Lead on electronic media and also placed in the SCM ClearCase repository.

11. Change Control

Change control is performed as follows:

11.1. Software modifications.

Problems shall be documented on an STR and brought before the SCCB for direction. If the SCCB decides the STR shall be resolved, the board chair shall direct SCM to open a Software Change Request (SCR) in the SCM Data Base (SCMDB). The SCR is assigned to a designated Developer who will examine the software for a probable correction set. After examination, the Developer will bring this recommendation before a subsequent SCCB. If approved, and if a change is required, the SCCB will direct SCM to check out designated files. These files will be placed in a subdirectory under the Developer's directory, and the SCMDB appropriately updated. Once the files have been updated, verified as part of an engineering build, and gone through the Peer Review process, the Developer shall return to the SCCB, who will direct SCM to include the files in the next SCM build.

SCM will check the files back in to ClearCase, update the SCMDB, and perform an SCM build. Following verification by the Test group, and by direction of the SCCB, the files will be "frozen" as a baseline within ClearCase; the executables will be delivered to the Project Lead; and the SCR and related STR will be closed within the SCMDB.

11.2. Documentation modifications.

Problems shall be documented on an STR and brought before the SCCB for direction. If the SCCB decides the STR shall be resolved, the board chair shall direct SCM to open a Software Change Request (SCR) in the SCM Data Base (SCMDB). The SCR is assigned to a designated Developer who will examine the documentation for a probable correction set. After examination, the Developer will bring this recommendation before a subsequent SCCB; if approved, and if a change is required, the SCCB will direct SCM to check out designated documents. These will be placed in a subdirectory under the Developer’s directory, and the SCMDB appropriately updated. Once the files have been updated and completed the Peer Review process, the Developer shall return to the SCCB, who will direct SCM to baseline the documents and release copies to all applicable members of the SDIP team.

12. Metrics

Metrics for SCM shall be collected according to the guidelines identified within the SDIP SDP.

13. Tools

SCM shall use the following tools.

- Rational ClearCase to control items.
- The (Microsoft Access) SCMDB to track SCRs/STRs through their life cycle.
- Build scripts, provided by Software Development personnel.
- Compilers/linkers, as specified in the SDIP SDP.

14. Schedule

SCM schedules shall be in compliance with schedules identified in the SDIP SDP.

15 Resources

Activity	**Estimated Hours**
Develop SCM Plan	40
Support Team Meetings	5 hours
Support SCCB meetings	1 hour
Baseline software and documentation	4 hours

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E – SOFTWARE TEST PLAN

Skinny Driver's Instrument Panel (SDIP) Test Plan

July10, 2002

US Army TACOM
Tank Automotive Research, Development, and Engineering Center
Next Generation Software Engineering Technology Area
AMSTA-TR-R/MS 265
Warren, MI 48397-5000

	Eric Jochum Author, Software Engineer	Date
Approval	Nadia Abadir Test Manager	Date
Approval	John Bohn Project Leader	Date
Approval	Christopher Ostrowski Systems Engineer	Date
Approval	Karen LaFond SQA Manager	Date
Final Approval For Use	Michael S. Saboe, Ph.D. Associate Director Next Generation Software Engineering Technology Area	Date

REVISION LOG

Version	Date	Author	Description
0.1	June 13, 2002	Eric Jochum	Initial Draft.
0.2	June 20, 2002	Eric Jochum	Applied changes generated from Peer Review held on June 19, 2002.
1.0	July 8, 2002	Eric Jochum	First release to be signed off after condition updates per the and placed under CM.

TABLE OF CONTENTS

1 TEST PLAN IDENTIFIER

SDIP-35-4-1

2 INTRODUCTION

2.1 Purpose

The purpose of this document is to describe the approach of the testing activities for the Skinny Driver's Instrument Panel (SDIP) project in the Next Generation Software Engineering Technology Area (NextGen). These complement and support the overall SDIP software development effort. This Test Plan shall be used in conjunction with the SDIP Test Coverage and Test Cases.
The Test Cases shall be used to perform regression testing as applicable. The objective is to create repeatable tests for use in proving out and demonstrating a system or group of systems.

2.2 Background

The SDIP Project's goal is to produce M1A2 simulation software, implementing a functionally reduced version of the Driver's Integrated Display (DID) from that vehicle. This modified DID is referred to as the Skinny Driver's Instrument Panel (SDIP). The software resulting from the SDIP Project is collectively known as the SDIP CSCI. The SDIP CSCI is intended to be made available to non-government entities for research purposes. The focus of this research will be the development and addition of software monitoring probes to U.S. Army weapons systems.

2.3 Scope

This plan is applicable to the SDIP project managed by the Next Generation Software Engineering Technology Area.

2.4 Referenced Documents

Title	Document Reference Number
Independent Software Test Procedure	NGSEL-35-3-5 1.0 7/19/01 Next Generation Software Engineering Technology Area, TARDEC
M1A2 PDSS Process and Procedures	Pdss_part3.doc 12/16/1998 NT-Share on 'ice':/szfranJ/PDSS Training
SDIP Software Development Plan	SDIP-22-4-1 2.0 TBA Next Generation Software Engineering Technology Area, TARDEC
SDIP Software Requirements Specification	SDIP-21-4-1 2.0 Next Generation Software Engineering Technology Area, TARDEC
Software Product Engineering Procedure	NGSEL-35-3-1 1.0 7/19/01 Next Generation Software Engineering Technology Area, TARDEC
Test Case Template	NGSEL-35-4-3 1.0 6/05/02 Next Generation Software Engineering Technology Area, TARDEC
Test Coverage Template	NGSEL-35-4-2 1.0 6/05/02 Next Generation Software Engineering Technology Area, TARDEC
Test Plan Template	NGSEL-35-4-1 1.0 6/05/02 Next Generation Software Engineering Technology Area, TARDEC
Verify Tests Procedure	NGSEL-25-3-3 1.0 4/27/99 Next Generation Software Engineering Technology Area, TARDEC

3 ROLES AND RESPONSIBILITIES

Roles	Responsibilities
Associate Director	• Approve the project Test Plan, Test Coverage, and Test Cases
Project Leader	• Approve the project Test Plan, Test Coverage, and Test Cases • Certify / signal product is ready for test • Review test results
SCM Manager	• Place the project Test Plan, Test Coverage, and Test Cases materials under CM • Provide CM-built software version for testing
SEPG Lead	• Approve any identified tailoring of the organization standard software process
SQA Engineer	• Witness software testing
SQA Manager	• Approve the project Test Plan, Test Coverage, and Test Cases • Review test results
Systems Engineer	• Approve the project Test Plan, Test Coverage, and Test Cases • Review test results
Test Manager	• Coordinate the preparation of the project Test Plan and Test Coverage • Approve the project Test Plan, Test Coverage, and Test Cases • Review test results
Test Staff	• Prepare the Test Cases and update the project Test Coverage • Perform the tests per the SDIP Test Cases and record the results

4 ACRONYMS AND DEFINITIONS

Acronym	Definition
API	Application Program Interface
CM	Configuration Management
CSC	Computer Software Component
CSCI	Computer Software Configuration Item. A logical collection of CSCs
DID	Drivers Integrated Display
Next Gen	Next Generation Software Engineering Technology Area
NGSEL	
SCM	Software Configuration Management
SCR	Software Change Request
SDIP	Skinny Driver's Instrument Panel
SDP	Software Development Plan
SEPG	Software Engineering Process Group
SQA	Software Quality Assurance
SRS	Software Requirement Specification
STR	System Trouble Report
TACOM	Tank Automotive and Armaments Command
TARDEC	Tank Automotive Research, Development, and Engineering Center
Test Case	One test procedure. Results are summarized in the SDIP Test Coverage
Test Coverage	SDIP summary information on product testing
Test Plan	Defines the SDIP test approach

5 TEST ITEMS

The test items are 3 components of the SDIP CSCI. The following components shall be tested:

- SDIP CSC
- 1553 API and emulation CSC
- Autopilot CSC

6 FEATURES TO BE TESTED

The following features will be tested:

- SDIP CSC
  - Simulates DID menu structure
  - Reacts as a DID to emulated 1553 packets
- 1553 API and emulation CSC
  - 'Project Critical' packets are maintained
  - 'System Status' packets
- Autopilot CSC
  - Collect and maintain waypoint list
  - Send emulated 1553 packets
  - Reacts to change of status packets

7 FEATURES NOT TO BE TESTED

Functions not in the scope of SDIP SRS will not be tested.

8 TEST PLAN ENTRY CRITERION

The basic entry criterion for the Test Plan is that the SDIP SRS has been approved.

9 TESTING INPUTS

The inputs to testing are:

- The SDIP schedule
- The SDIP SRS
- The SDIP Test Plan
- The SDIP Test Cases
- CM Build of SDIP Code

9.1 Environmental Needs

The minimum equipment, operating systems, communications software, hardware and firmware needed to perform the testing are specified in the SDIP SDP.

9.2 Schedule

The SDIP Schedule is an input to this Test Plan. The Testing Schedule is reflected in the overall Project schedule.

10 TESTING OUTPUTS / DELIVERABLES

The outputs from the SDIP testing activities are:

10.1 Outputs from Test Planning

- Test Plan
- Test Coverage / Report
- Test Cases

10.2 Outputs from Testing

- Updated Test Plan
- Updated project schedule
- As encountered, STR(s) per the SDIP SDP
- Updated Test Coverage / Report
- Updated Test Cases with test results

11 EXIT CRITERIA

11.1 Planning Phase

The exit criteria for the planning phase is that:

- SDIP Test Plan is reviewed and signed off
- The Test Coverage fields listed below have been completed for project, the Test Coverage is reviewed and signed off
  - The header rows
  - "Requirements" column
  - "Test Cases" column
- The Test Cases fields listed below have been completed for the project, the Test Cases reviewed and signed off
  - "Objective"
  - "Pass/Fail"
  - "Equipment Needed to Perform Test"
  - "Pre-Test Procedures"
  - "Test Procedure"
  - "Test Case Version"
  - "Test Case Revised By"
  - Test Case Revised by "Date"
  - "Test Case Approved By
  - Test Case Approved by "Date"

11.2 Testing Execution

The exit criteria for the test executing phase is that:

- All of the fields in the project's Test Coverage have been completed, including the "Results" columns, the document reviewed, and signed off
- All of the fields in the project's Test Cases have been completed, the document reviewed, and signed off
- If required, this Test Plan is updated, reviewed, and signed off

12 PROCEDURE

Developed prior to the start of testing, the SDIP Test Coverage shall document the summary information for the SDIP Test Cases. It will contain the traceability between the SDIP requirements and the individual Test Cases. When testing has been completed, the Test Coverage shall be completed with the summary information on the time it took to perform the testing, whether or not rework was involved (by the number of iterations before a given Test Case passed), and when the test results were reviewed. Should regression testing be required, the SDIP prior completed Test Coverage shall be used to identify which Test Cases to include in the regression testing sequence and to project the estimated time to complete the regression testing.

12.1 Define the Test Methodology/ Approach

The levels and types of testing which shall be performed for this project are as follows:

- SDIP CSCI

  Using the SCM controlled Test Coverage template, the Test Manager shall complete an initial draft of the Test Coverage. The test location shall be Next Gen Lab. The SDIP test schedule shall be included as part of the SDIP master schedule.

- Independent testing (refer to NGSEL-35-3-5 Independent Software Test Procedure)
  Independent testing shall use the formats defined by the Test Coverage (NGSEL-35-4-2) and Test Case (NGSEL-35-4-3). Independent testing includes, but is not limited to, the following:
  - Black box functional testing, which includes:
    - Boundary value testing
    - Robustness testing
    - Worst case analysis testing
    - Special value testing
    - Multiple features testing
  - The types of testing include the following:
    - Set-up / initialization testing
    - Communications protocol testing
    - Feature testing

12.2 Develop Test Cases

Once the types of testing have been identified in the SDIP Test Coverage, the Test Staff shall perform analysis on the requirement to determine the test cases. Using the SCM controlled Test Case template, the Test Staff shall document each test case. This shall include identifying:

- The test hardware and software configurations
- The initialization and set up procedure and data
- The global test environment for each test
- The test suspension criteria
- The test results recording methods and applicable storage
- The expected test results
- If applicable, the build cycle verification checklist

The Test Staff may create more than one Test Case to meet the SDIP project requirements. For example, all of the tests may have the same shared test set up procedure, with individual tests covering communication protocols, specific features, etc. The Test Staff shall update the SDIP Test Coverage as needed.

12.3 Conduct Peer Review

The SDIP Test Plan, Test Coverage, and Test Cases shall be formally reviewed and signed off per the Next Gen NGSEL-37-3-1 Peer Review Procedure prior to the start of the testing using the SDIP Test Cases. Any action items arising from the review shall be addressed. Please refer to the SDIP master schedule for the planned peer review scheduling.

12.4 Conduct Testing

The Test Staff shall perform the testing per the documented Test Cases and shall record the results in the appropriate positions on the documents. The Test Staff shall record the Test Plan summary information on the SDIP Test Coverage. Should there be any errors encountered during the testing, the Test Staff shall write STRs; as per M1A2 PDSS part 3, to document the conditions under which the errors were encountered.

12.5 Post-Verification Activities

The Test Staff shall review the test results and metrics with the Test Manager, Systems Engineer, Project Leader, and the SQA Manager. The materials shall be put under SCM control. Any STRs created during the testing shall be tracked to closure.

12.6 Regression Testing

Regression testing will be conducted if applicable for this project.

13 STAFFING AND TRAINING NEEDS

13.1 Staffing

The following staff is needed to carry out this SDIP testing activities:

Role	Quantity
Test Manager	1
Test Staff	1

13.2 Training

The Test staff shall be trained by a member of the SDIP project team to operate the SDIP CSCI.

14 RISKS AND CONTINGENCIES

Refer to the SDIP SDP.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Luqi
   Naval Postgraduate School
   Monterey, California

4. Mikhail Auguston
   Naval Postgraduate School
   Monterey, California

5. Christopher D. Miles
   United States Army TACOM
   Warren, Michigan

6. Chris Ostrowski
   United States Army TACOM
   Warren, Michigan

7. Douglas Gersky
   United States Army TACOM
   Warren, Michigan

8. John Bohn
   United States Army TACOM
   Warren, Michigan

9. Joe Szafranski
   United States Army TACOM
   Warren, Michigan